L Number	Hits	Search Text	DB	Time stamp
6	1	6170014.pn.	USPAT	2004/01/20 11:33
7	1	6170014.pn. and advantag$	USPAT	2004/01/20 11:11
8	289	firmware WITH disk WITH controller	USPAT	2004/01/20 11:26
9	89	(firmware WITH disk WITH controller) and network	USPAT	2004/01/20 11:13
10	9	(firmware WITH disk WITH controller) same advantag$	USPAT	2004/01/20 11:14
11	9	(firmware WITH disk WITH controller) same advantag$	USPAT	2004/01/20 11:14
12	1	delay with server WITH disk WITH controller	USPAT	2004/01/20 11:34
13	0	monitor with bootup with attempt	USPAT	2004/01/20 11:27
14	0	check with bootup with attempt	USPAT	2004/01/20 11:27
15	0	check with (boot near up) with attempt	USPAT	2004/01/20 11:27
16	0	monitor with (boot near up) with attempt	USPAT	2004/01/20 11:27
17	1	(delay with server WITH disk WITH controller) and advantag$	USPAT	2004/01/20 11:33
18	1	6119162.pn.	USPAT	2004/01/20 11:34
19	0	6119162.pn. and delay	USPAT	2004/01/20 11:34
20	255	delay WITH disk WITH controller	USPAT	2004/01/20 11:34
21	1	(delay WITH disk WITH controller) same server	USPAT	2004/01/20 11:34
22	34	(delay WITH disk WITH controller) and server	USPAT	2004/01/20 11:36
23	1	(delay WITH disk WITH controller) and (server with content)	USPAT	2004/01/20 11:37
24	12	(delay WITH disk WITH controller) and (server with connect$)	USPAT	2004/01/20 11:44
25	1	5604873.pn.	USPAT	2004/01/20 11:44
26	1	5604873.pn. and advantag$	USPAT	2004/01/20 11:46
27	0	5604873.pn. and sever	USPAT	2004/01/20 11:46
28	1	5604873.pn. and server	USPAT	2004/01/20 11:47
29	1	5604873.pn. and server same delay	USPAT	2004/01/20 14:22
30	5040	watson.in.	USPAT	2004/01/20 14:23
31	1	watson.in. and (boot near up)	USPAT	2004/01/20 14:24
32	1	5,475,839.pn.	USPAT	2004/01/20 14:24
33	502	monitor$ WITH boot	USPAT	2004/01/20 14:27
34	274	(monitor$ WITH boot) and network	USPAT	2004/01/20 14:26
35	32	(monitor$ WITH boot) same advantag$	USPAT	2004/01/20 14:26
36	17	((monitor$ WITH boot) same advantag$) and network	USPAT	2004/01/20 14:26
37	20	monitor$ WITH boot WITH number	USPAT	2004/01/20 14:36
38	1	5654905.pn.	USPAT	2004/01/20 15:08
39	1	5654905.pn. and advantag$	USPAT	2004/01/20 15:08

40	0	(monitor$ WITH boot WITH number) same advantag$	USPAT	2004/01/20 14:36
41	17	(monitor$ WITH boot WITH number) and advantag$	USPAT	2004/01/20 14:39
42	27	advertisement WITH boot	USPAT	2004/01/20 14:41
43	0	(advertisement WITH boot) same server	USPAT	2004/01/20 14:41
44	12	(advertisement WITH boot) and server	USPAT	2004/01/20 14:47
45	1	6373498.pn.	USPAT	2004/01/20 14:47
46	1	6373498.pn. and advantag$	USPAT	2004/01/20 14:49
47	1	6373498.pn. and boot	USPAT	2004/01/20 14:51
48	1	6373498.pn. and advertisement	USPAT	2004/01/20 14:56
49	0	firmware WITH delay$ WITH event WITH threshold	USPAT	2004/01/20 14:57
50	2151599	firmware WITH delay$ connect$	USPAT	2004/01/20 14:57
51	20799	firmware WITH delay$ connect$ WITH server	USPAT	2004/01/20 14:58
52	513	monitor$ WITH event WITH threshold	USPAT	2004/01/20 14:58
53	25	(firmware WITH delay$ connect$ WITH server) and (monitor$ WITH event WITH threshold)	USPAT	2004/01/20 15:01
54	25	(firmware WITH delay$ connect$ WITH server) and (monitor$ WITH event WITH threshold)	USPAT	2004/01/20 15:01
55	13	((firmware WITH delay$ connect$ WITH server) and (monitor$ WITH event WITH threshold)) and delay$	USPAT	2004/01/20 15:01
56	1	5696701.pn.	USPAT	2004/01/20 15:08
57	1	5696701.pn. and advantag$	USPAT	2004/01/20 15:08
62	2	((disk near drive) WITH protected WITH area) same advantag$	USPAT	2004/01/20 15:37
61	30	(disk near drive) WITH protected WITH area	USPAT	2004/01/20 15:39
63	10	((disk near drive) WITH protected WITH area) and network	USPAT	2004/01/20 15:38
64	6	store WITH (disk near drive) WITH protected WITH area	USPAT	2004/01/20 16:40
65	1	6546489.pn.	USPAT	2004/01/20 16:11
66	1	6546489.pn. and advantag$	USPAT	2004/01/20 16:11
67	1	6119162.pn.	USPAT	2004/01/20 16:40
68	1	6119162.pn. and address	USPAT	2004/01/20 16:40
-	1147	713/201.ccls.	USPAT	2004/01/15 14:02
-	503	713/201.cor.	USPAT	2004/01/15 14:02
-	0	eneboe-in.	USPAT	2004/01/15 14:03
-	6	eneboe near michael	USPAT	2004/01/15 14:04
-	0	(eneboe near michael) and server	USPAT	2004/01/15 14:11
-	680	(select$ or choos$) with (network near address)	USPAT	2004/01/15 14:12
-	0	((select$ or choos$) with (network near address)) and ((select$ or choos$) with server WITH contact$ WITH program)	USPAT	2004/01/15 14:56

-	14	(select$ or choos$) with server WITH contact$ WITH program	USPAT	2004/01/15 14:55
-	371	(select$ or choos$) with ISP	USPAT	2004/01/15 14:21
-	10	((select$ or choos$) with (network near address)) and ((select$ or choos$) with ISP)	USPAT	2004/01/20 15:36
-	1	(((select$ or choos$) with (network near address)) and ((select$ or choos$) with ISP)) and firmware	USPAT	2004/01/15 14:35
-	3	((select$ or choos$) with server WITH contact$ WITH program) and firmware	USPAT	2004/01/15 14:54
-	316	(select$ or choos$) with communicator	USPAT	2004/01/15 14:55
-	0	((select$ or choos$) with (network near address)) and (((select$ or choos$) with communicator) same server)	USPAT	2004/01/15 14:56
-	30	((select$ or choos$) with communicator) same server	USPAT	2004/01/15 15:46
-	1	6119162.pn.	USPAT	2004/01/15 15:46
-	0	6119162.pn. and disk	USPAT	2004/01/15 15:50
-	830	content near server	USPAT	2004/01/15 15:50
-	15	(content near server) same (user near (id or password))	USPAT	2004/01/15 15:52
-	3810	content with time with interval	USPAT	2004/01/15 15:54
-	1	((content near server) same (user near (id or password))) and (content with time with interval)	USPAT	2004/01/15 15:54
-	94966	content with time	USPAT	2004/01/15 15:54
-	6	((content near server) same (user near (id or password))) and (content with time)	USPAT	2004/01/15 15:54

US005604873A

United States Patent [19]

Fite et al.

[11] Patent Number: 5,604,873

[45] Date of Patent: Feb. 18, 1997

[54] **CIRCUITRY FOR CONTROLLING POWER APPLICATION TO A HOT DOCKING SCSI SCA DISK DRIVE**

[75] Inventors: **Robert J. Fite**, Hillsboro; **David A. Underwood**, Scappoose; **Dale Rembold**, Boring, all of Oreg.

[73] Assignee: **Intel Corporation**, Santa Clara, Calif.

[21] Appl. No.: **366,230**

[22] Filed: **Dec. 28, 1994**

[51] **Int. Cl.**[6] **H01J 13/00**

[52] **U.S. Cl.** **395/283**; 395/281; 395/282; 395/750; 395/733

[58] **Field of Search** 395/750, 281, 395/282, 283, 700, 425, 800, 894; 369/75.1; 364/708; 371/10.1; 360/39

[56] **References Cited**

U.S. PATENT DOCUMENTS

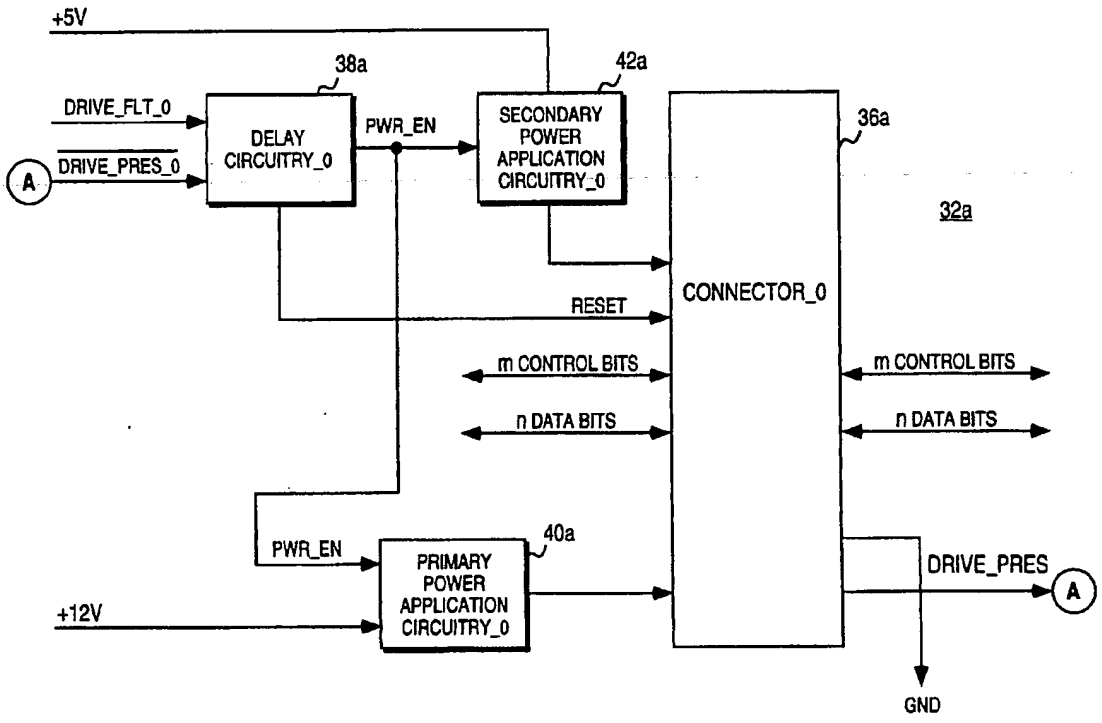
4,999,787	3/1991	McNally et al.	364/514
5,077,722	12/1991	Geist et al.	369/75.1
5,297,067	3/1994	Blackborow et al.	364/708
5,303,244	12/1994	Watson	371/10.1
5,317,697	4/1994	Husak et al.	395/283
5,386,567	1/1995	Lien et al.	395/700
5,454,080	9/1995	Fasig et al.	395/283
5,488,572	1/1996	Belmont	364/514 R

Primary Examiner—Jack B. Harvey
Assistant Examiner—Raymond N. Phan
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A disk drive controller having a plurality of disk drive interfaces, each interface includes a connector, a delay circuit, and a set of power application circuits is provided to a server to support hot docking of SCA drives. Each connector is adapted to mate with a hot docking disk drive having equal length connecting pins, and detect the presence of such disk drive when the hot docking disk drive makes contact with the connector. Each delay circuit generates a set of properly delayed enabling signals to the corresponding power application circuits. Each set of power application circuits regulates power applications to the hot docking disk drive making contact with the corresponding connector. The delayed and regulated manner of applying power prevents voltage and power swings that might disrupt on-going operations and/or cause damages to the neighboring drives.

24 Claims, 6 Drawing Sheets



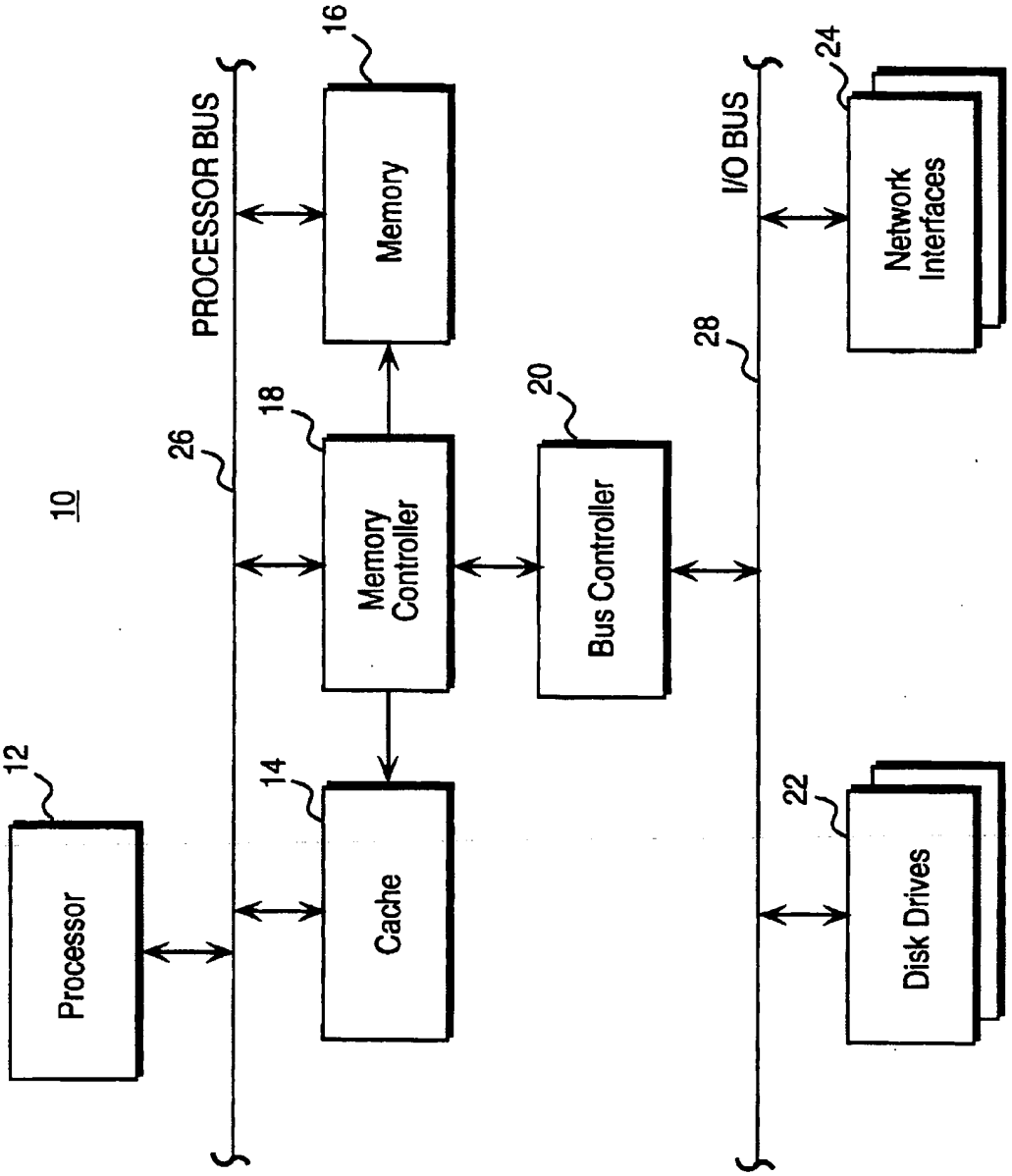

10
PROCESSOR BUS
I/O BUS
Processor
12
Cache
14
Memory
16
Memory Controller
18
Bus Controller
20
Disk Drives
22
Network Interfaces
24
26
28

Figure 1

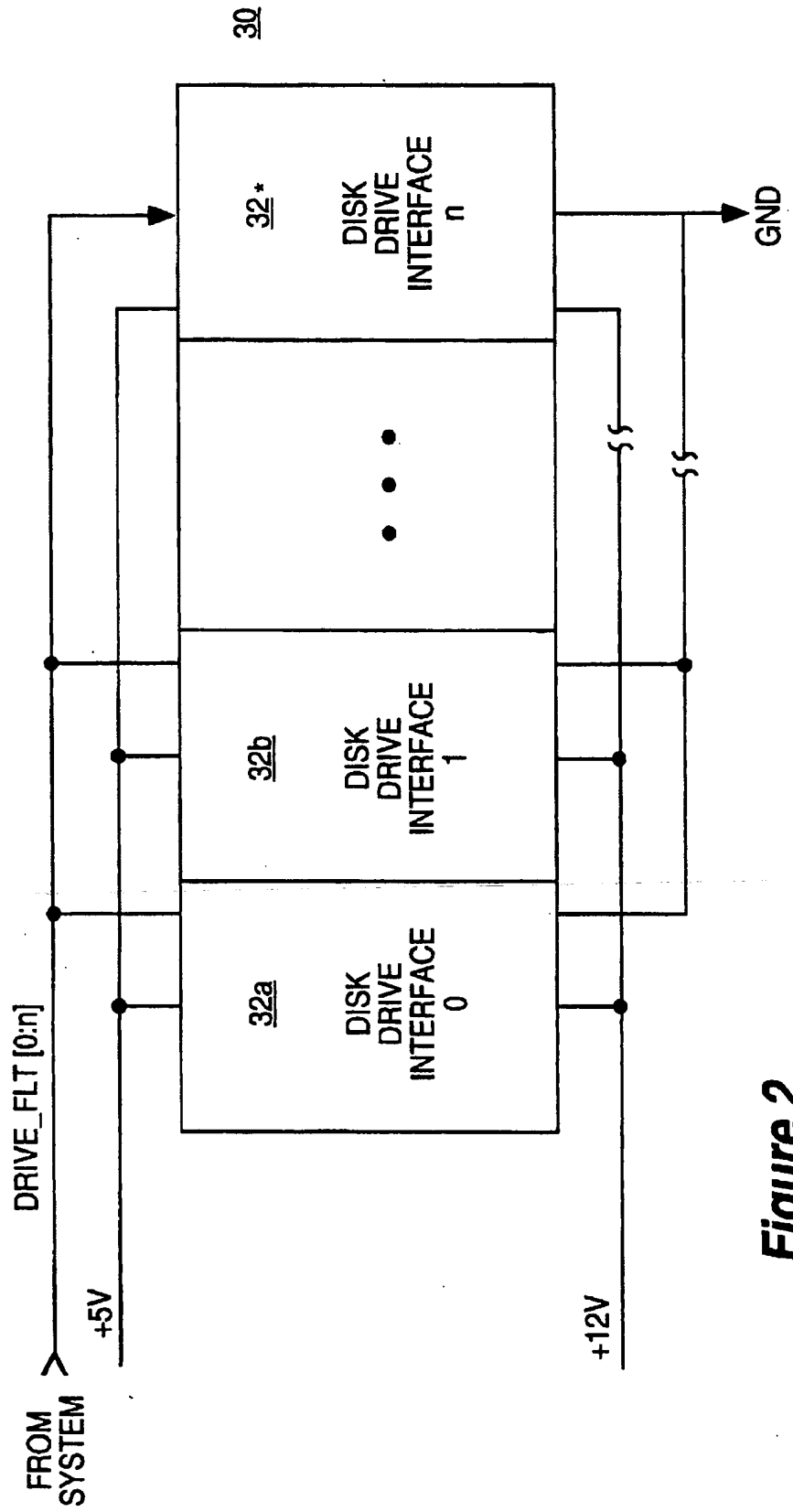

30
32*
DISK DRIVE INTERFACE n
32b
DISK DRIVE INTERFACE 1
32a
DISK DRIVE INTERFACE 0
GND
DRIVE_FLT [0:n]
+5V
+12V
FROM SYSTEM

Figure 2

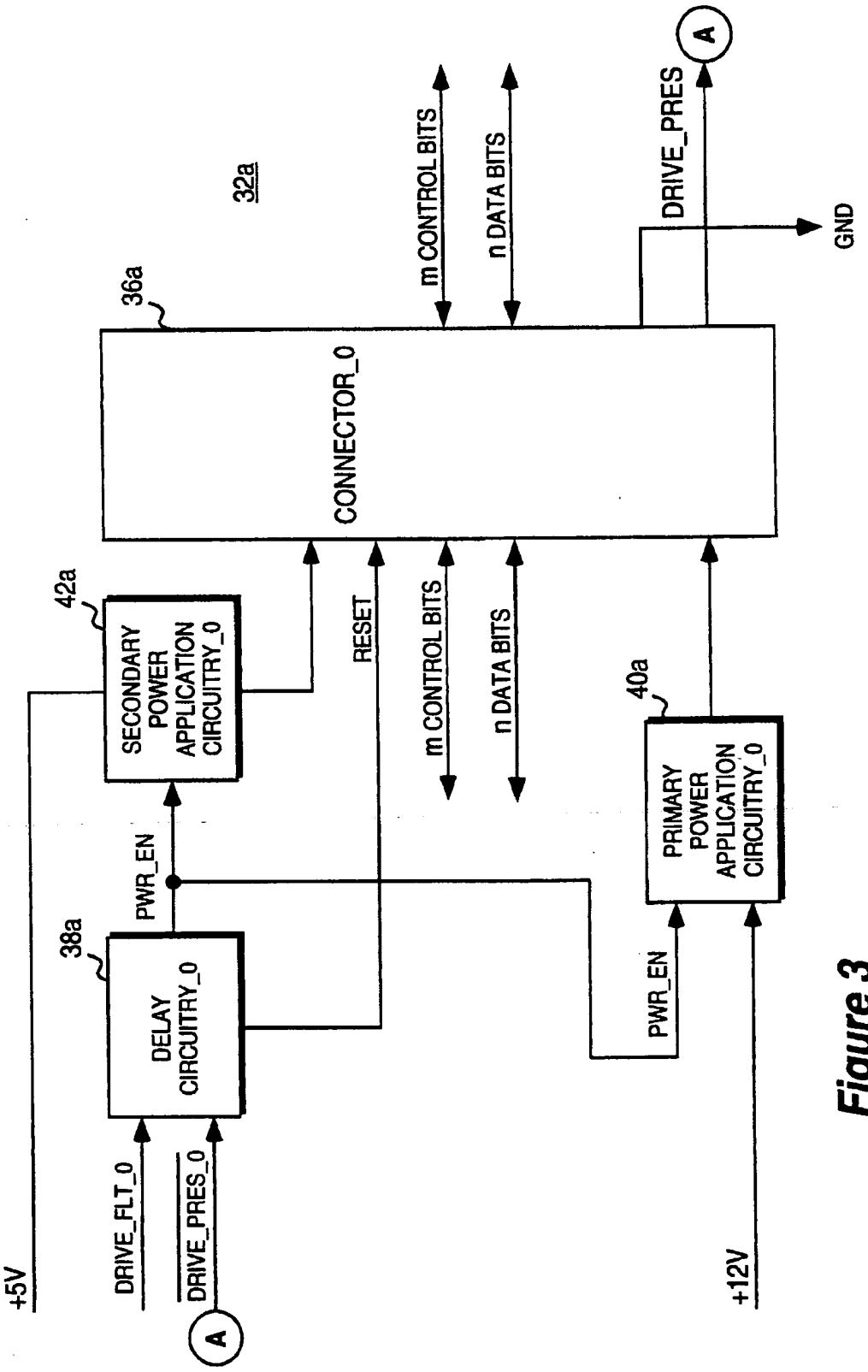

32a
36a
CONNECTOR_0
m CONTROL BITS
n DATA BITS
DRIVE_PRES
GND
A
42a
SECONDARY POWER APPLICATION CIRCUITRY_0
RESET
m CONTROL BITS
n DATA BITS
40a
PRIMARY POWER APPLICATION CIRCUITRY_0
PWR_EN
38a
DELAY CIRCUITRY_0
PWR_EN
DRIVE_FLT_0
DRIVE_PRES_0
+5V
+12V
A

Figure 3

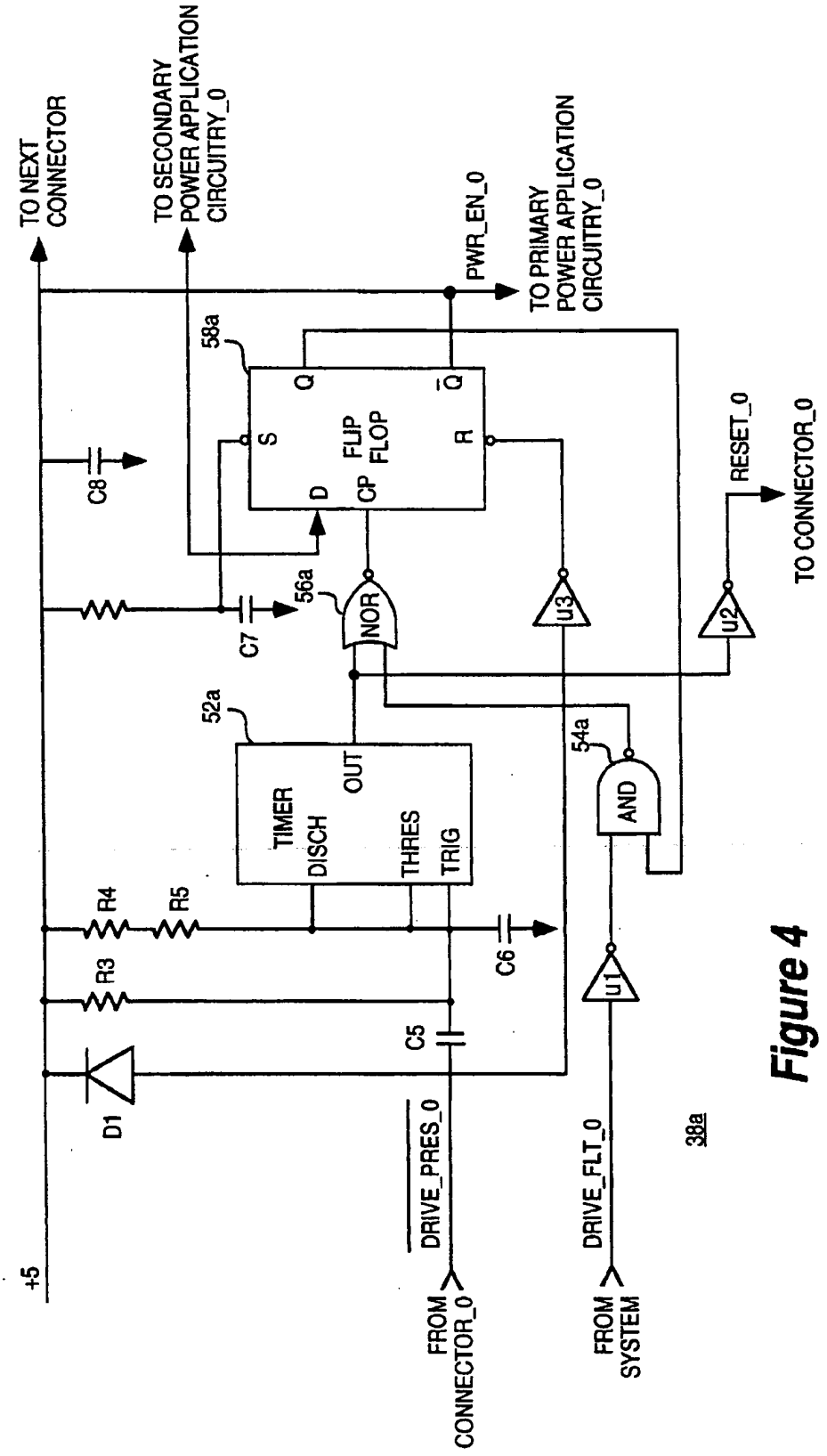

TO NEXT CONNECTOR
TO SECONDARY POWER APPLICATION CIRCUITRY_0
PWR_EN_0
TO PRIMARY POWER APPLICATION CIRCUITRY_0
58a
Q
$\bar{Q}$
S
R
D
CP
FLIP FLOP
C8
RESET_0
TO CONNECTOR_0
56a
NOR
C7
U3
U2
52a
54a
OUT
TIMER
DISCH
THRES
TRIG
AND
R4
R5
C6
R3
U1
C5
D1
DRIVE_PRES_0
DRIVE_FLT_0
38a
+5
FROM CONNECTOR_0
FROM SYSTEM

Figure 4

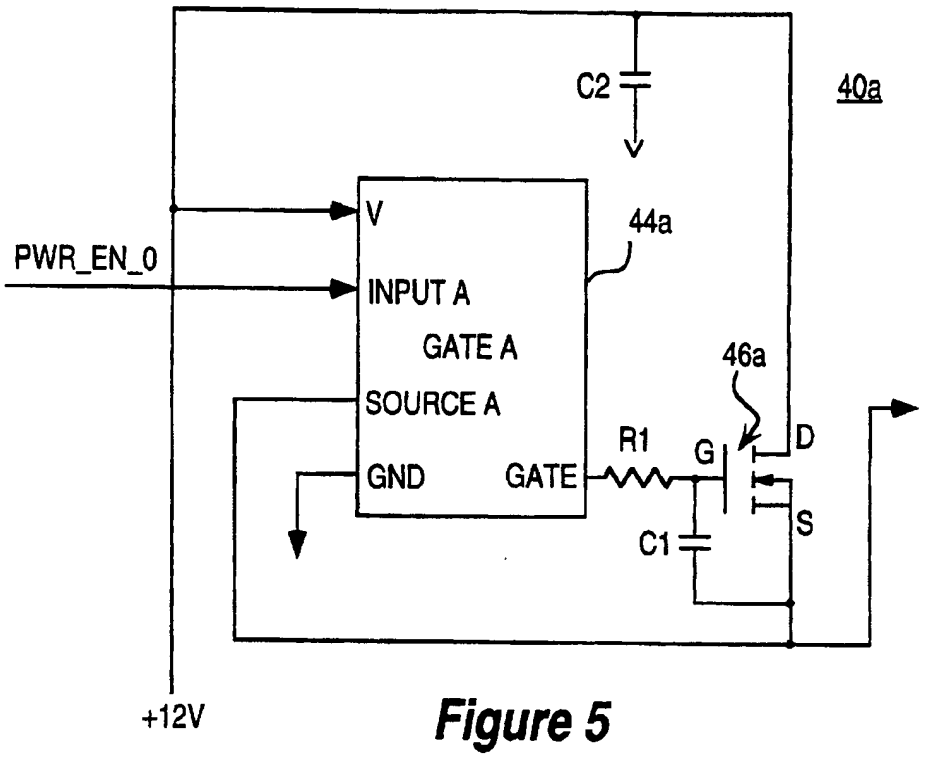


Figure 5

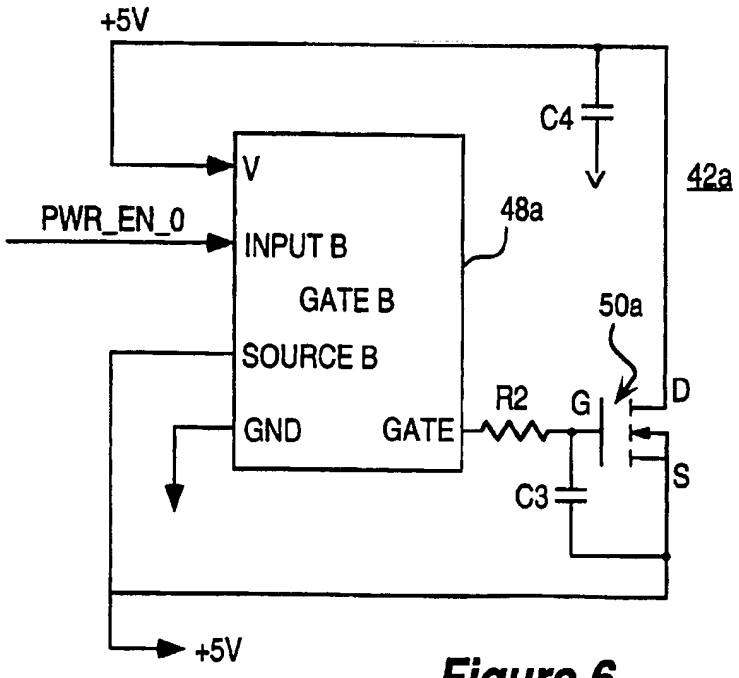
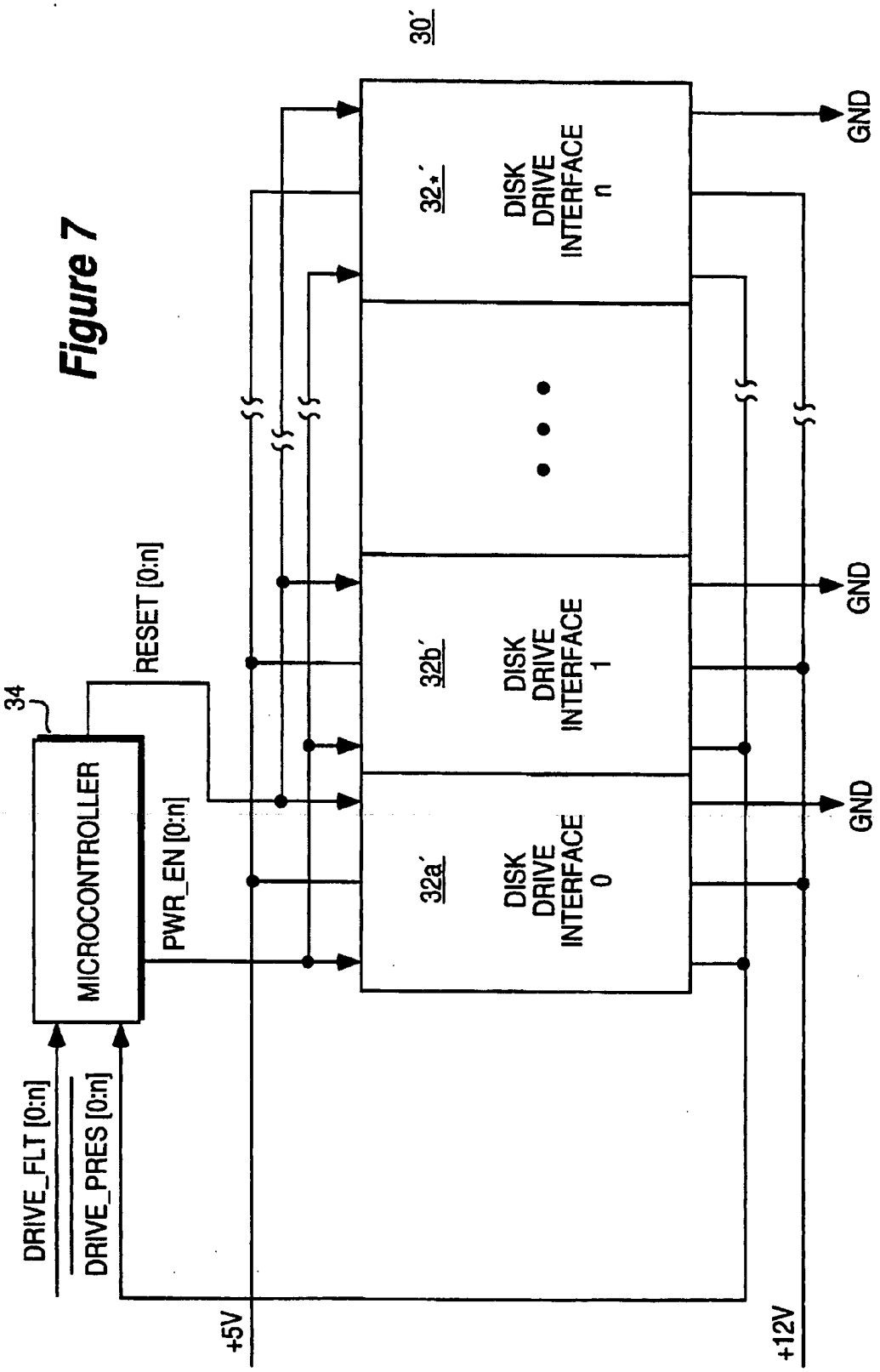


Figure 6

Figure 7


30'
32a'
DISK DRIVE INTERFACE 0
32b'
DISK DRIVE INTERFACE 1
32x'
DISK DRIVE INTERFACE n
GND
GND
GND
34
MICROCONTROLLER
RESET [0:n]
PWR_EN [0:n]
DRIVE_FLT [0:n]
DRIVE_PRES [0:n]
+5V
+12V

CIRCUITRY FOR CONTROLLING POWER APPLICATION TO A HOT DOCKING SCSI SCA DISK DRIVE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the fields of disk drives and computer systems. In particular, the present invention relates to controlling power application when hot docking a Small Computer System Interface (SCSI) Single Connector Attachment (SCA) disk drive.

2. Background Information

As more and more microprocessor based servers are employed in various critical or sensitive business or scientific applications, the expectation of their reliability and availability increases in tandem. A particular aspect that is of increasing interest is the availability and reliability of the server's disk drives. In today's servers, it is not uncommon to find a not insignificant number of disk drives storing many gigabytes of data. As a result, hot swappable or hot dockable disk drives have emerged as a high priority feature.

Hot swapping or hot docking refers to the ability to remove a malfunctioning disk drive from the server and reinsert a properly functioning disk drive into the server without having to halt and/or otherwise shut the server down. Excessive power and voltage swings must be prevented when removing the malfunctioning disk drive and reinserting the replacement drive, to prevent disruption to operations and/or physical damages to the neighboring drives. The conventional approach is to employ disk drives having connecting pins that are of uneven lengths. More specifically, special 5 v and 12 v pins having lengths that are longer than associated pins are used to make preliminary electrical contacts with the power supply of the server, thereby stabilizing the power and voltage of the server, before the shorter operational pins would make full electrical contact with the power supply of the server.

An SCA drive by definition has equal length connecting pins. Thus, traditionally SCA drives are not considered to be hot swappable or hot dockable. Since SCA drives, due to other reasons, are a lot more economical than most of the hot dockable drives employed today, it is desirable to be able to hot dock SCA drives.

As will be disclosed in more detail below, the circuitry of the present invention controls power application to a hot docking SCA disk drive, thereby allowing the SCA disk drive to be hot dockable. These and other advantages will be evident to those skilled in art from the detailed descriptions to follow.

SUMMARY AND OBJECTS OF THE INVENTION

The desirable results are advantageously achieved by providing a disk drive controller having a plurality of disk drive interfaces, each interface includes a connector, a delay circuit and a set of power application circuits. Each connector is adapted to mate with a docking disk drive having equal length connecting pins, and to report the presence of such disk drive making contact with the connector. The delay circuit is used to generate a set of properly delayed enabling signals to allow time for the connector to go from partial engagement to full engagement. The delayed enabling signals are generated responsive to the reported disk drive presence. Each set of power application circuits are used to apply power to the docking disk drive making contact with the corresponding connectors, in a regulated manner, to prevent in rush of current due to changing loads. The power is applied responsive to the delayed enabling signals. The delayed and regulated manner of applying power prevents voltage and power swings that might disrupt on-going operations and/or cause damages to the neighboring drives.

In one embodiment, the delay circuit comprises primarily a delay timer coupled to the corresponding connector and a power supply, and a flip flop serially coupled to delay timer. In one variation of this embodiment, the delay circuit further comprises a number of Boolean gates complementing the delay timer and the flip flop for factoring the fault state of the corresponding drive in generating the set of properly delayed enabling signals. In an alternate embodiment, a micro-controller is employed to generate the set of properly delayed enabling signals for the various sets of power application circuits.

In one embodiment, each set of power application circuits includes a primary power application circuit for applying power to the docking disk drive making contact with the corresponding connector via a +12 v line, and a secondary power application circuit for applying power to the same corresponding connector via a +5 v line. Both power application circuits are equipped with the ability to apply the power in a gradual manner. In a particular variation of this embodiment, the primary and secondary power application circuits are similarly constituted. Each circuit comprises primarily of a gate coupled to the power input and a FET transistor serially coupled to the gate. Preferably, each power application circuit is further provided with another parallel connection to the FET transistor to serve as a by-pass to compensate for the step load requirement of the docking disk drive.

BRIEF DESCRIPTION OF DRAWINGS

The present invention will be described by way of exemplary embodiments but not limitations illustrated in the accompanying drawings in which like references denote similar elements, and in which:

FIG. 1 illustrates an exemplary server incorporating the teachings of the present invention;

FIG. 2 illustrates the relevant portions of one embodiment of the disk drive controller of the present invention employed by the exemplary server of FIG. 1;

FIG. 3 illustrates one embodiment of one of the disk drive interfaces of FIG. 2 in further detail;

FIG. 4 illustrates one embodiment of the delay circuitry of FIG. 3 in further detail;

FIGS. 5–6 illustrate one embodiment each of the primary and secondary power application circuitry of FIG. 3 in further detail;

FIG. 7 illustrates the relevant portions of an alternate embodiment of the disk drive controller of the present invention employed by the exemplary server of FIG. 1.

DETAILED DESCRIPTION OF THE INVENTION

In the following description, for purposes of explanation, specific numbers, materials and configurations are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well known systems are

shown in diagrammatic or block diagram form in order not to obscure the present invention.

Referring now to FIG. 1, a block diagram illustrating an exemplary computer server incorporating the teachings of the present invention. The computer server 10 comprises processor 12, cache memory 14, main memory 16, and memory controller 18 coupled to each other via a processor bus 26. The computer server 10 further comprises bus controller 20, disk drives 22 and network interfaces 24 coupled to each other via input/output (I/O) bus 28. Memory and bus controllers 18 and 20 are also coupled to each other. The disk drives 22 are SCA drives and include a SCA disk drive controller incorporated with the teachings of the present invention, enabling the SCA disk drives 22 to be hot dockable.

Before describing the present invention in further detail, it should be noted that while the present invention is being described in the context of the above described exemplary computer server 10, the present invention may be practiced in computer servers having different system architectures. In fact, based on the descriptions to follow, it will be appreciated that the present invention may also be practiced on desktop computers, notebook computers and the like.

Referring now to FIG. 2, which illustrates the relevant portions of one embodiment of the SCA disk controller 30 in further detail. Disk drive controller 30 includes disk drive interfaces 32*a*–32* incorporated with the teachings of the present invention. Disk drive interfaces 32*a*–32* are identical interfaces serially coupled to each other and receiving power supply via a set of various voltage lines. For the illustrated embodiment, disk drive interfaces 32*a*–32* further receive drive fault states (DRIVE_FLT [0:n]) as inputs. Drive fault states denote whether faults are detected by the server 10 for the various disk drives 22; and they are set and reset by the server 10.

In one embodiment, disk drive controller 30 includes eight (8) such disk drive interfaces 32*a*–32*h*. All eight interfaces 32*a*–32*h* receive a primary power input via a +12 v line and a secondary power input via a +5 v line. In this embodiment, all eight interfaces 32*a*–32*h* also receive their corresponding fault states (DRIVE_FLT [0:8]) as inputs.

Referring now to FIG. 3, one embodiment of first disk drive interface 32*a* is illustrated in further detail. As illustrated, first disk drive interface 32*a* comprises a connector 36*a*, delay circuit 38*a* and a set of power application circuits 40*a*–42*a*. For the illustrated embodiment, the set of power application circuits 40*a*–42*a* includes a primary power application circuit 40*a* and a secondary power application circuit 42*a*.

Connector 36*a* is adapted to mate with a docking disk drive having equal length connecting pins, and detect the presence of such disk drive (DRIVE_PRES_0) when the disk drive makes contact with the connector 36*a*. Delay circuit 38*a* generates the properly delayed enabling signals (PWR_EN_0) for the power application circuits 40*a*–42*a*, allowing time for the connector 36*a* to go from initial partial engagement to full engagement. Enabling signal PWR_EN_0 is generated responsive to the detection signal (DRIVE_PRES_0) factoring in the drive's fault state (DRIVE_FLT_0). Primary and secondary power application circuits 40*a*–42*a* are used to regulate a first and a second power application to connector 36*a* via a +12 v and a +5 v line, preventing in rush of currents as a result of changing load. Both primary and secondary power are applied gradually responsive to properly delayed enabling signals PWR_EN_0.

The amount of time delays required to allow the connector 36*a* to go from partial engagement to full engagement is dependent on the physical characteristics of the disk driver carrier. In one embodiment, the required delay time is about ¼ sec. The rate of power application is dependent on the frequency responsiveness of the power supplies. In one embodiment, the rate of power application is about 500 usec. The delayed and regulated manner of applying power prevent power and voltage swings that might interrupt on going operations or cause damages to the neighboring disk drives.

Referring now to FIG. 4, which illustrates one embodiment of delay circuit 38*a* in further details. As illustrated, delay circuit 38*a* primarily comprises of a delay timer 52*a* coupled to the corresponding connector and a power supply, and a flip flop 58*a* serially couple to the delay timer 52*a*. More specifically, the discharge and threshold terminals of delay timer 52*a* is coupled to the power supply of the +5 v line via resistors R4 and R5, and the trigger terminal of delay timer 52*a* is coupled to the corresponding connector. Flip flop 58*a* is serially coupled to the output terminal of delay timer 52*a*. Additionally, for the illustrated embodiment, AND gate 54*a* and NOR gate 56*a* are provided to complement delay timer 52*a* and flip flop 58*a* to allow the drive's fault state to be factored in the generation of the properly delayed enabling signal (PWR_EN_0).

Delay circuit 38*a* outputs active PWR_EN_0 upon receiving the complement of DRIVE_PRES_0 denoting the presence of the corresponding drive, applying proper amount of delay and ensuring the fault state (DRIVE_FLT_0) of the corresponding drive has been reset by the server. The various resistor and capacitor values are empirically determined. In one embodiment, the values of R4 and R5 are 2M and 1M ohms respectively.

Referring now to FIGS. 5–6, which illustrate one embodiment each of primary and secondary power application circuits 40*a*–42*a* in further details. As illustrated, primary and secondary power application circuits 40*a*–42*a* are similarly constituted. Each circuit 40*a* or 42*a* comprises primarily of gate 44*a* or 48*a* coupled to the power supply via either the +12 v or the +5 v line, and FET transistor 46*a* or 50*a* serially coupled to gate 44*a* or 48*a*. More specifically, the output terminal of gate 44*a* or 48*a* is serially coupled to the gate input terminal of FET transistor 46*a* or 50*a* through resistor R1 or R2. Preferably, each circuit 40*a* or 42*a* further includes another parallel connection between the power supply on the +12 v or +5 v line and FET transistor 46*a* or 50*a* having C2 and C4 coupled to ground as shown. More specifically, the parallel connections are between the power supply of the +12 v and +5 v lines and the drains of FET transistors 46*a* and 50*a* respectively.

Gate 44*a* or 48*a* prevents the power supplied via the +12 v or +5 v to be applied, unless the properly delayed enabling signal (PWR_EN_0) is active. If PWR_EN_0 is active, power supplied via the +12 v and +5 v lines are applied through FET transistors 46*a* and 50*a* respectively. However, because of R1 and C1, and R2 and C3, the power from the +12 v and +5 v lines are applied gradually. The parallel connections (induding C2 and C4) serve as by-passes to the FET transistors 46*a* and 50*a* for compensating the step load requirement of the disk drive.

The values for R1, R2, and C1–C4 are empirically determined. In one embodiment, the values of R1 and R2 are 1M ohms and 300K ohms respectively, whereas the values of C1–C4 are all 0.01 uF.

Referring now to FIG. 7, an alternate embodiment of disk drive controller 30' is illustrated. Disk drive controller 30'

similarly comprises disk drive interfaces 32a'–32*'. However, in lieu of providing a delay circuit to each of the disk drive interfaces 32a'–32*', micro-controller 34 is provided instead. DRIVE_PRES [0:n] and DRIVE_FLT [0:n] are all routed to micro-controller 34. In response, micro-controller 34 generates PWR_EN [0:n] as described earlier.

Thus, a circuit for controlling power application to a hot docking SCSI SCA disk drive is described. While the circuit of the present invention has been described in terms of the illustrated embodiments, those skilled in the art will recognize that the invention is not limited to the embodiments described. The present invention can be practiced with modification and alteration within the spirit and scope of the appended claims. The description is thus to be regarded as illustrative instead of restrictive on the present invention.

What is claimed is:

1. A disk drive interface comprising:

a) a connector interface for mating with a hot docking disk drive having equal length connecting pins and detecting the hot docking disk drive's presence when the hot docking disk drive's equal length connecting pins make contact with the connector interface; and

b) a first power application circuit coupled to the connector interface for applying a first power in a time delayed and gradual manner to the hot docking disk drive making contact with the connector interface, the first power being applied in the time delayed and gradual manner in response to receiving a first time delayed enabling signal which is provided to the first power application circuit as a result the connector interface detecting the presence of the hot docking disk drive.

2. The disk drive interface as set forth in claim 1 wherein the first power application circuit applies power supply from a +12 v line in the time delayed and gradual manner to the hot docking disk drive making contact with the connector interface.

3. The disk drive interface as set forth in claim 1 wherein the disk drive interface further comprises

c) a delay circuit coupled to the connector interface and the first power application circuit for receiving a detection signal denoting the presence of the hot docking disk drive from the connector interface, and in response, delaying for a predetermined amount of time, and then providing the first properly time delayed enabling signal to the first power application circuit.

4. The disk drive interface as set forth in claim 1 wherein the disk drive interface further comprises

c) a second power application circuit coupled to the connector interface for applying a second power in a time delayed and gradual manner to the hot docking disk drive making contact with the connector interface, the second power being applied in the time delayed manner in response to receiving a second time delayed enabling signal which is provided to the second power application circuit as a result the connector interface detecting the presence of the hot docking disk drive.

5. The disk drive interface as set forth in claim 1 wherein the disk drive interface further comprises:

d) a delay circuit coupled to the connector interface, the first and the second power application circuits for receiving a detection signal denoting the presence of the hot docking disk drive from the connector interface, and in response, delaying for a predetermined amount of time, and then providing the first and the second properly time delayed enabling signals to the first and the second power application circuits respectively.

6. The disk drive interface as set forth in claim 3 wherein the delay circuit further factors into consideration a fault state of the hot docking disk drive in providing the first properly time delayed enabling signal.

7. The disk drive interface as set forth in claim 4 wherein the second power application circuit applies power supply from a +5 v line in a time delayed and gradual manner to the hot docking disk drive making contact with the connector interface.

8. The disk drive interface as set forth in claim 5 wherein the delay circuit further factors into consideration a fault state of the hot docking disk drive in providing the first and the second properly time delayed enabling signals.

9. A disk drive controller comprising:

a) a plurality of disk drive interfaces, each disk drive interface having

a.1) a connector interface for mating with a hot docking disk drive having equal length connecting pins and detecting the hot docking disk drive's presence when the hot docking disk drive's equal length connecting pins make contact with the connector interface, and

a.2) a first power application circuit coupled to the connector interface for applying a first power in a time delayed and gradual manner to the hot docking disk drive making contact with the connector interface, the first power being applied in the time delayed and gradual manner in response to receiving a first time delayed enabling signal which is provided to the first power application circuit as a result the connector interface detecting the presence of the hot docking disk drive.

10. The disk drive controller as set forth in claim 9 wherein each disk drive interface further comprises:

a.3) a delay circuit coupled to the connector interface and the first power application circuit for receiving a detection signal denoting the presence of the hot docking disk drive from the connector interface, and in response, delaying for a predetermined amount of time, and then providing the first properly time delayed enabling signal to the first power application circuit.

11. The disk drive controller as set forth in claim 9 wherein each disk drive interface further comprises

a.3) a second power application circuit coupled to the connector interface for applying a second power in a time delayed and gradual manner to the hot docking disk drive making contact with the connector interface, the second power being applied in the time delayed manner in response to receiving a second time delayed enabling signal provided as a result the connector interface detecting the presence of the hot docking disk drive.

12. The disk drive controller as set forth in claim 9, wherein the disk drive controller further comprises:

b) a microcontroller coupled to the plurality of disk drive interfaces for receiving a signal denoting presence of a disk drive at the connector interface from each of the disk drive interface, and in response, delaying for a predetermined amount of time, and providing each of the disk drive interfaces with a first properly time delayed enabling signal.

13. The disk drive controller as set forth in claim 11 wherein each disk drive interface further comprises:

a.4) a delay circuit coupled to the connector interface, the first and the second power application circuits for receiving a detection signal denoting the presence of the hot docking disk drive from the connector inter-

face, and in response, delaying for a predetermined amount of time, and then providing the first and the second properly time delayed enabling signals to the first and the second power application circuits respectively.

14. The disk drive controller as set forth in claim 12 wherein the microcontroller further receives a fault state of the docked disk drive for each disk drive interface, and factors the fault state into providing the corresponding first properly time delayed enabling signal.

15. The disk drive controller as set forth in claim 12 wherein each disk drive controller further comprises

c) a second power application circuit coupled to the connector interface for applying a second power in a time delayed and gradual manner to the hot docking disk drive making contact with the connector interface, the second power being applied in the time delayed manner in response to receiving a second time delayed enabling signal which is provided to the second power application circuit as a result the connector interface detecting the presence of the hot docking disk drive.

16. The disk drive controller as set forth in claim 15 wherein the microcontroller further receives a fault state of the docked disk drive for each disk drive interface, and factors the fault state into providing the corresponding first and second properly time delayed enabling signals.

17. A computer system comprising:

a) a plurality of disk drives; and

b) a disk drive controller adapted for hot docking the disk drives, induding

b.1) a plurality of disk drive interfaces, each disk drive interface having

b.1.1) a connector interface for mating with a hot docking disk drive having equal length connecting pins and detecting the hot docking disk drive's presence when the hot docking disk drive's equal length connecting pins make contact with the connector interface, and

b.1.2) a first power application circuit coupled to the connector interface for applying a first power in a time delayed and gradual manner to the hot docking disk drive making contact with the connector interface, the first power being applied in the time delayed and gradual manner in response to receiving a first time delayed enabling signal which is provided to the first power application circuit as a result the connector interface detecting the presence of the hot docking disk drive.

18. The computer system as set forth in claim 17 wherein each disk drive interface further comprises:

b.1.3) a delay circuit coupled to the connector interface and the first power application circuit for receiving a detection signal denoting the presence of the hot docking disk drive from the connector interface, and in response, delaying for a predetermined amount of time, and then providing the first properly time delayed enabling signal to the first power application circuit.

19. The computer system as set forth in claim 17 wherein each disk drive interface further comprises

b.1.3) a second power application circuit coupled to the connector interface for applying a second power in a time delayed and gradual manner to the hot docking disk drive making contact with the connector interface, the second power being applied in the time delayed and gradual manner in response to receiving a second time delayed enabling signal which is provided to the second power application circuit as a result the connector interface detecting the presence of the hot docking disk drive.

20. The computer system as set forth in claim 17, wherein the disk drive controller further comprises:

b.2) a microcontroller coupled to the plurality of disk drive interfaces for receiving a signal denoting presence of a disk drive at the connector interface from each of the disk drive interface, and in response, delaying for a predetermined amount of time, and providing each of the disk drive interfaces with a first properly time delayed enabling signal.

21. The computer system as set forth in claim 17 wherein each disk drive interface further comprises

b.1.3) a second power application circuit coupled to the connector interface for applying a second power in a time delayed and gradual manner to the hot docking disk drive making contact with the connector interface, the second power being applied in the time delayed and gradual manner in response to receiving a second time delayed enabling signal which is provided to the second power application circuit as a result the connector interface detecting the presence of the hot docking disk drive.

22. The computer system as set forth in claim 19 wherein each disk drive interface further comprises:

b.1.4) a delay circuit coupled to the connector interface, the first and the second power application circuits for receiving a detection signal denoting the presence of the hot docking disk drive from the connector interface, and in response, delaying for a predetermined amount of time, and then providing the first and the second properly time delayed enabling signals to the first and the second power application circuits respectively.

23. The computer system as set forth in claim 20 wherein the microcontroller further receives a fault state of the docked disk drive for each disk drive interface, and factors the fault state into providing the corresponding first properly time delayed enabling signal.

24. The computer system as set forth in claim 21 wherein the microcontroller further receives a fault state of the docked disk drive for each disk drive interface, and factors the fault state into providing the corresponding first and second properly time delayed enabling signals.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,604,873

DATED : February 18, 1997

INVENTOR(S) : Fite et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In column 4 at line 59 delete "induding" and insert --including--

In column 7 at line 31 delete "induding" and insert --including--

Signed and Sealed this

Fifteenth Day of July, 1997

Bruce Lehman

Attest:

BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

US005654905A

United States Patent [19]

Mulholland et al.

[11] Patent Number: 5,654,905

[45] Date of Patent: Aug. 5, 1997

[54] **SYSTEM FOR TRACKING COMPUTER USAGE TIME**

[75] Inventors: **Candace Mulholland**, Foothill Ranch; **Jason T. Kurashige**, Irvine, both of Calif.

[73] Assignee: **AST Research, Inc.**, Irvine, Calif.

[21] Appl. No.: **528,945**

[22] Filed: **Sep. 15, 1995**

[51] **Int. Cl.**[6] **G06F 1/14**

[52] **U.S. Cl.** **364/569**; 364/281.3; 395/557

[58] **Field of Search** 364/552, 569, 364/280, 281.3, 264, 286.4, 925.2; 371/29.1; 395/650, 700, 550, 557-750

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,390,324 2/1995 Burckhartt et al. 395/375

5,557,784 9/1996 Dayan et al. 395/550

Primary Examiner—James P. Trammell

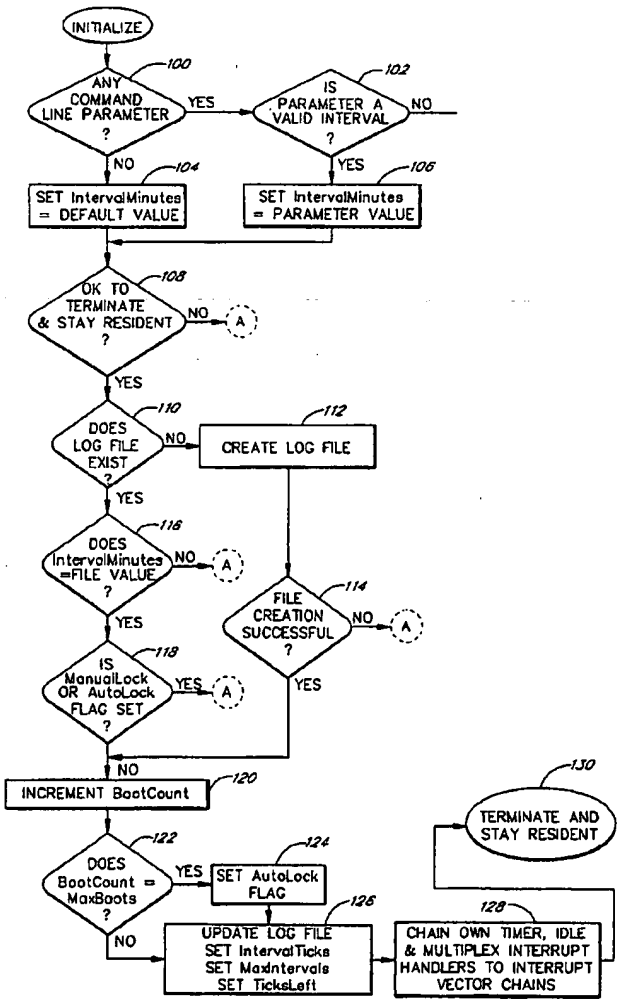
Assistant Examiner—Craig Steven Miller

Attorney, Agent, or Firm—Knobbe, Martens, Olson & Bear, LLP

[57] **ABSTRACT**

A system tracks the amount of time a computer system has operated, after its delivery to a new user. The system includes a terminate and stay resident (TSR) program which is loaded each time the computer is booted. The program counts system timer ticks and, at predetermined intervals but only when the operating system is idle, logs the amount of time counted in a log file located on the computer's local hard drive.

4 Claims, 6 Drawing Sheets



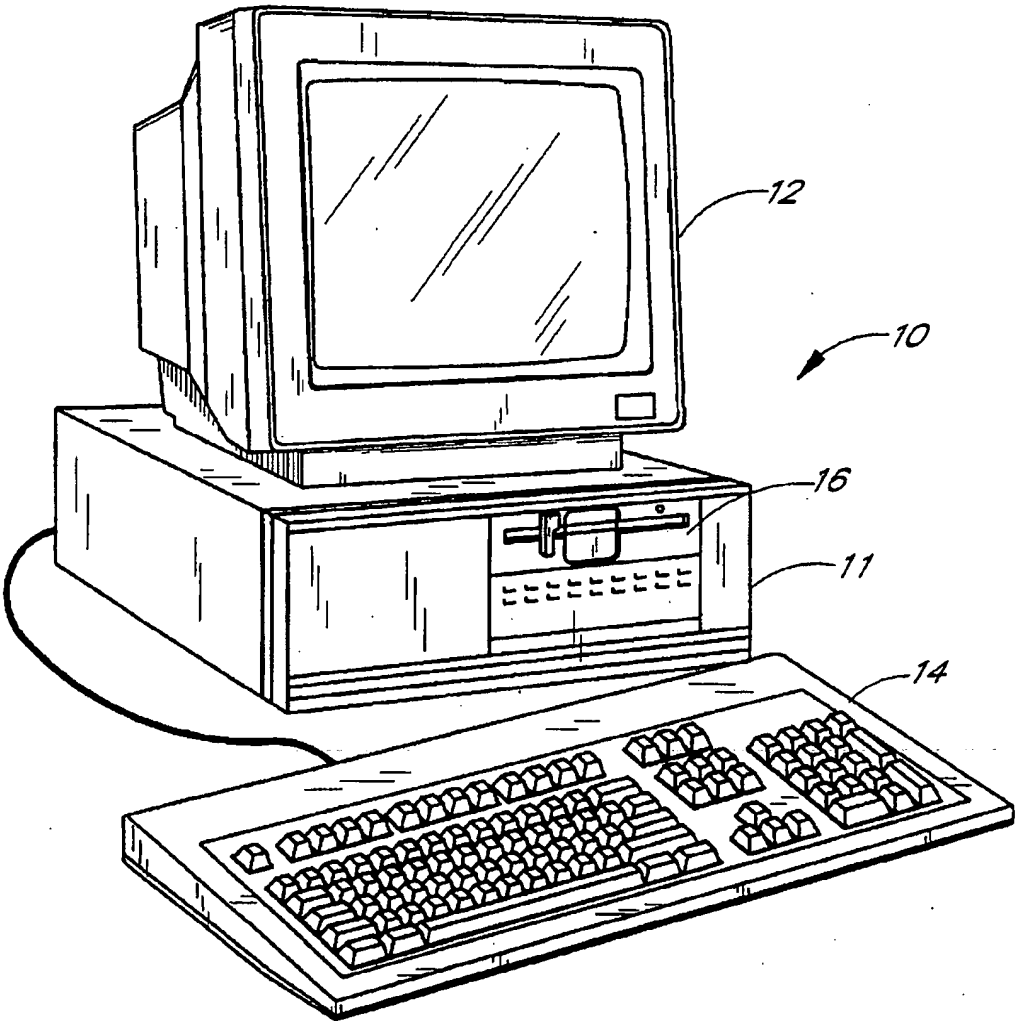


FIG. 1

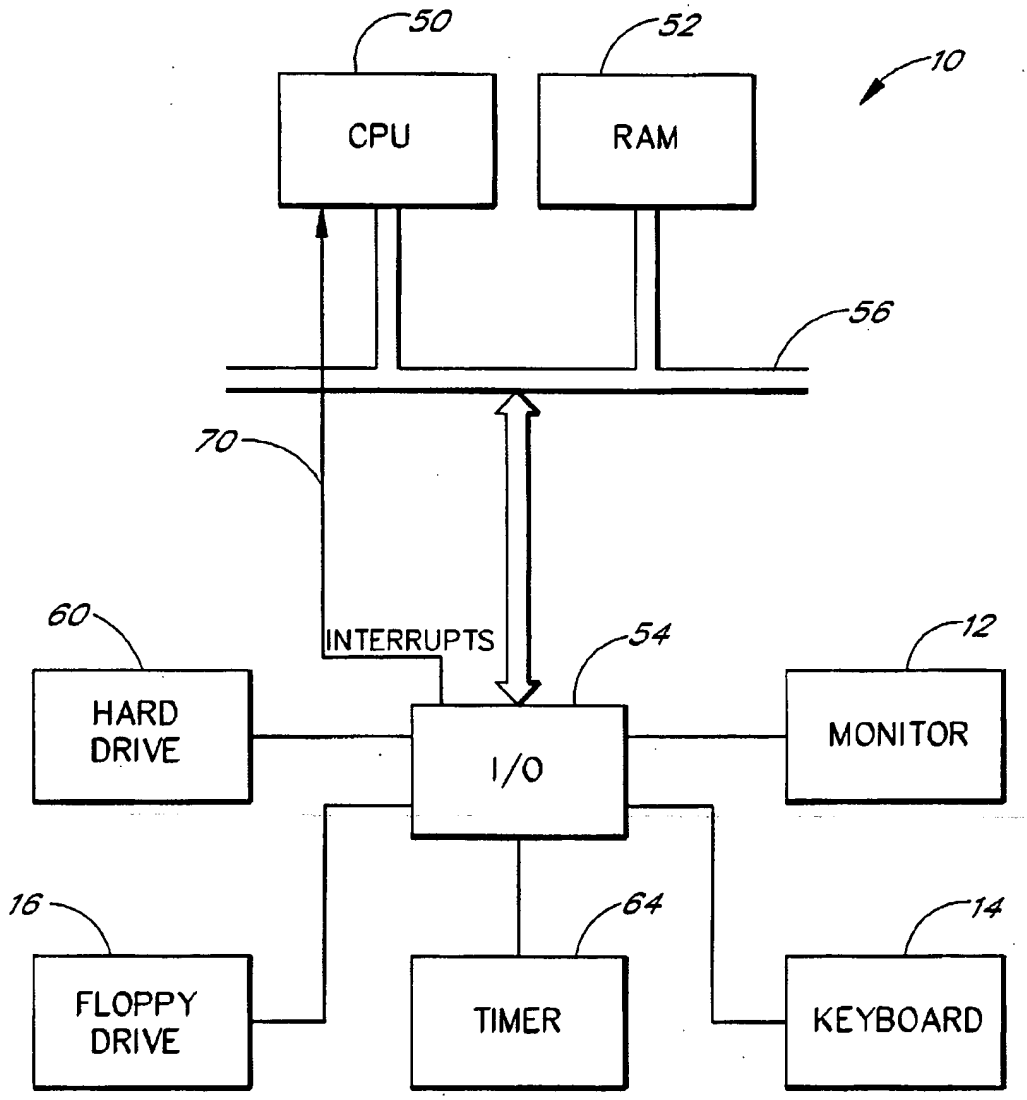

50
52
10
CPU
RAM
56
70
60
INTERRUPTS
54
12
HARD DRIVE
I/O
MONITOR
16
64
14
FLOPPY DRIVE
TIMER
KEYBOARD

FIG. 2

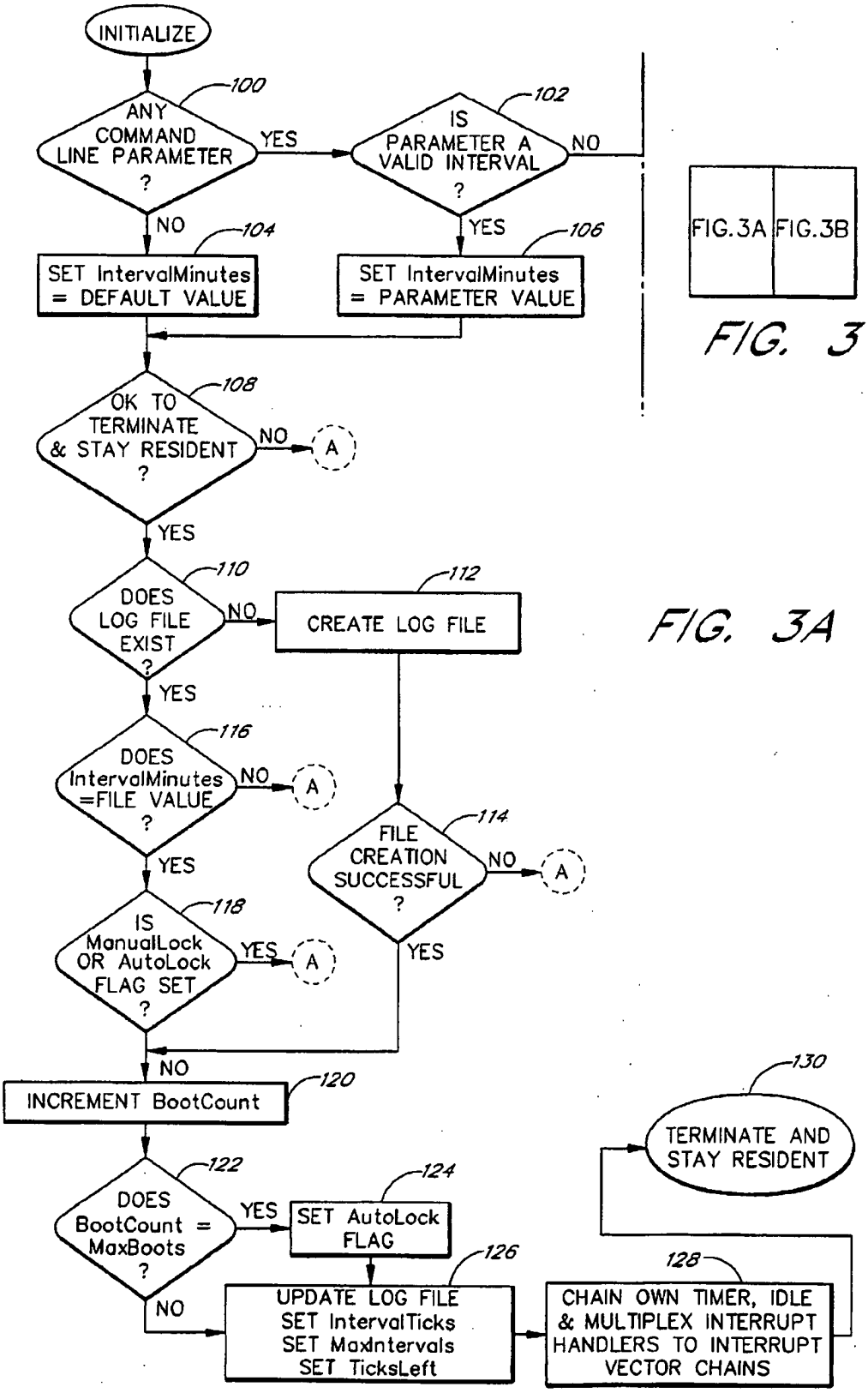


FIG. 3A FIG. 3B

FIG. 3

FIG. 3A

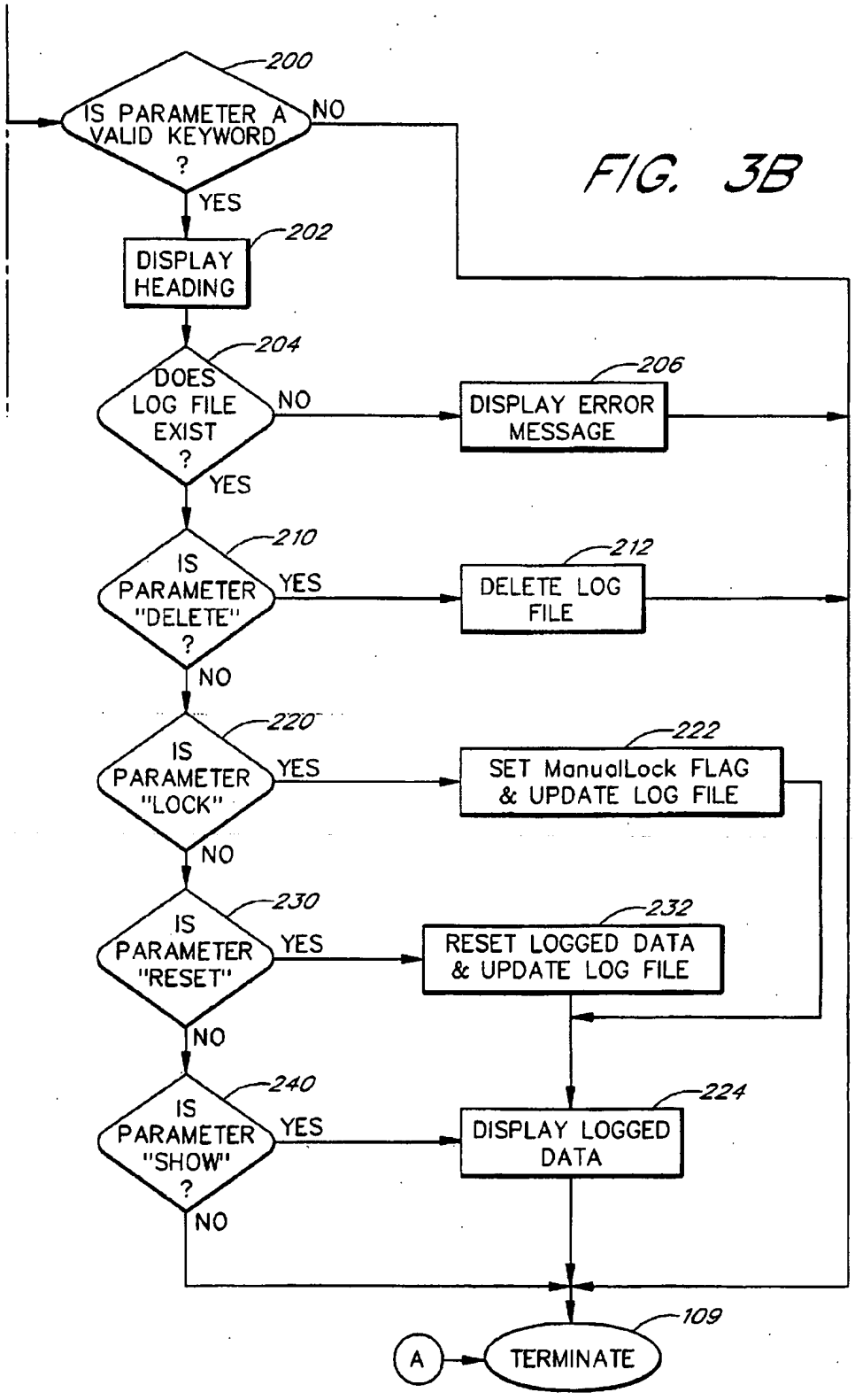

200
IS PARAMETER A VALID KEYWORD ?
NO
YES
202
DISPLAY HEADING
204
DOES LOG FILE EXIST ?
NO
206
DISPLAY ERROR MESSAGE
YES
210
IS PARAMETER "DELETE" ?
YES
212
DELETE LOG FILE
NO
220
IS PARAMETER "LOCK"
YES
222
SET ManualLock FLAG & UPDATE LOG FILE
NO
230
IS PARAMETER "RESET"
YES
232
RESET LOGGED DATA & UPDATE LOG FILE
NO
240
IS PARAMETER "SHOW" ?
YES
224
DISPLAY LOGGED DATA
NO
109
A
TERMINATE

FIG. 3B

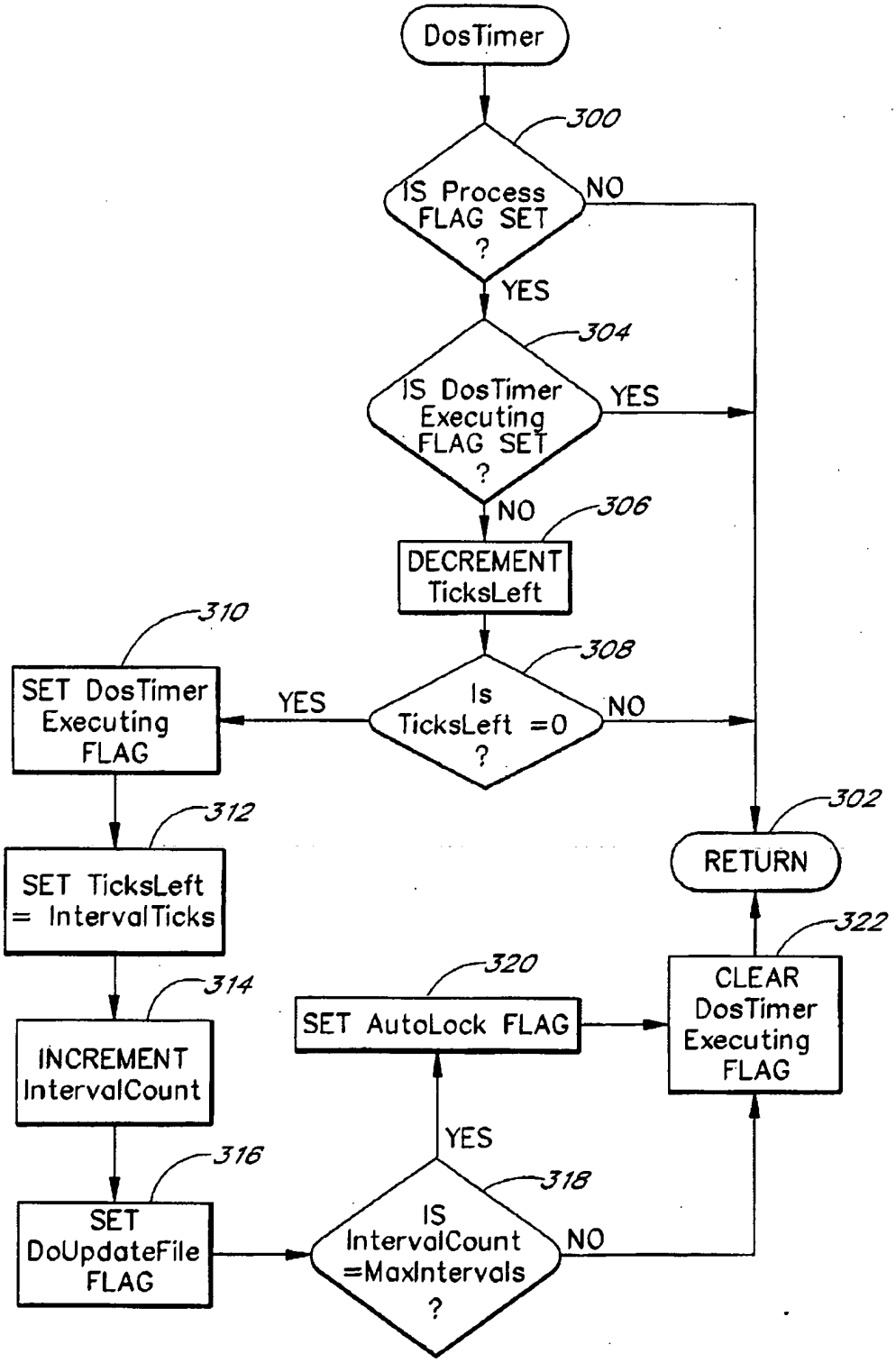

DosTimer
300
IS Process FLAG SET ?
NO
YES
304
IS DosTimer Executing FLAG SET ?
YES
NO
306
DECREMENT TicksLeft
308
Is TicksLeft =0 ?
YES
NO
310
SET DosTimer Executing FLAG
312
SET TicksLeft = IntervalTicks
314
INCREMENT IntervalCount
316
SET DoUpdateFile FLAG
318
IS IntervalCount =MaxIntervals ?
YES
NO
320
SET AutoLock FLAG
302
RETURN
322
CLEAR DosTimer Executing FLAG

FIG. 4

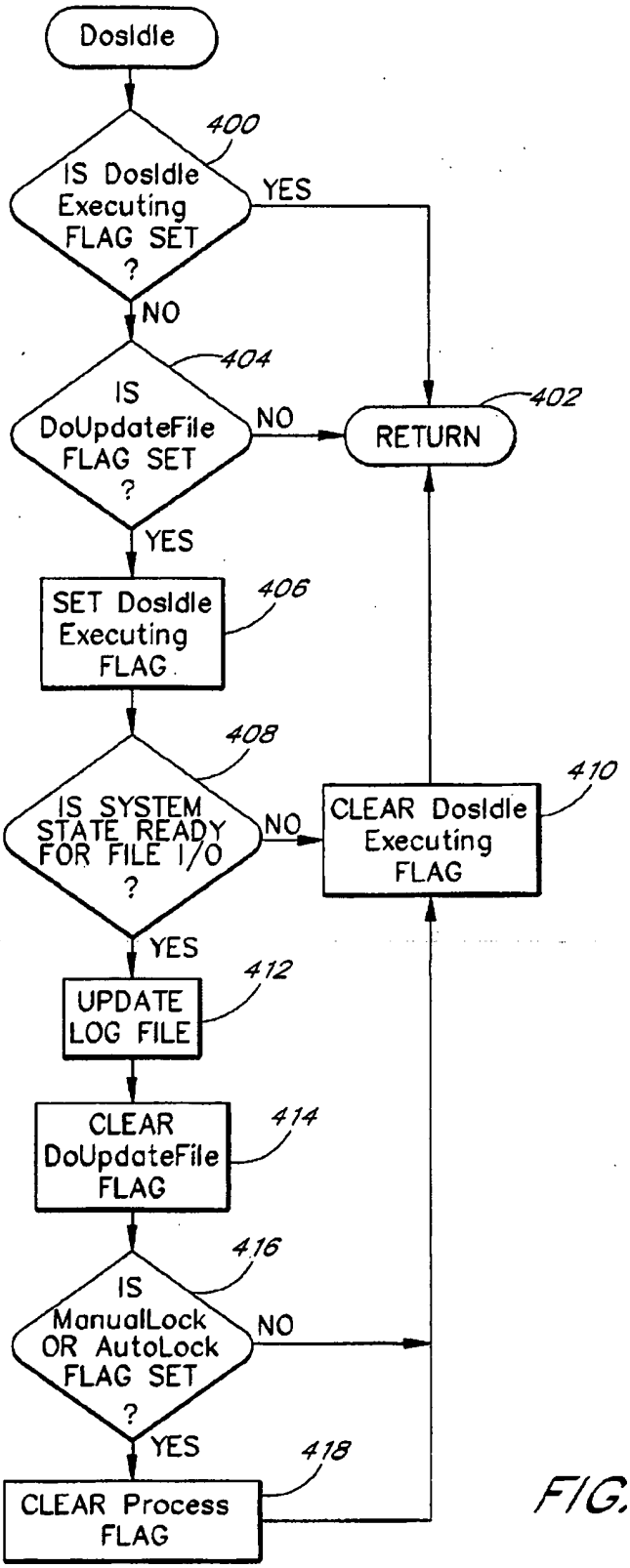

DosIdle
400
IS DosIdle Executing FLAG SET ?
YES
NO
404
IS DoUpdateFile FLAG SET ?
NO
402
RETURN
YES
SET DosIdle Executing FLAG
406
408
IS SYSTEM STATE READY FOR FILE I/O ?
NO
410
CLEAR DosIdle Executing FLAG
YES
412
UPDATE LOG FILE
CLEAR DoUpdateFile FLAG
414
416
IS ManualLock OR AutoLock FLAG SET ?
NO
YES
418
CLEAR Process FLAG

FIG. 5

SYSTEM FOR TRACKING COMPUTER USAGE TIME

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to systems for tracking the amount of time a computer has been operating. More specifically, this invention relates to background executing programs that track the amount of time a computer has operated after it has been delivered to a new user.

2. Description of Related Art

Computer sellers are sometimes asked to accept returned computer systems. Because the seller has no way of knowing how much the otherwise new system has been used, it is often forced to resell the computer as a used system, which of course commands a lower price. Were there some method of tracking the amount of time that a new computer system has been operated before its return, the seller could resell returned systems that were used infrequently or not at all for a much higher price.

There is thus a need for a method or system for tracking the amount of time a new computer system has been operated, after it has been delivered to a new user.

SUMMARY OF THE INVENTION

The present invention is a system for tracking the amount of time a computer has been operated, after the computer has been delivered to an end user. The system comprises a time logging program which is loaded into the computer's memory, and which is activated in a background mode each time the computer is booted. The time logging program counts timer interrupts issued by the computer's timer, and updates a log file located on the computer's hard drive at predetermined intervals. The log contains data representing the amount of time the computer has been on. Because the program operates in background mode, its execution is virtually transparent to the user, thus providing an elegant and unobtrusive method for tracking the amount of time the computer has been used. In the preferred embodiment, the time logging program is a DOS-based Terminate and Stay Resident (TSR) program.

In accordance with another aspect of the present invention, the time logging program intercepts idle interrupts, which are issued by the computer's operating system whenever it is idle. The time logging program updates the log file only during these idle periods, thus further ensuring that the program's operation does not interfere with the computer's other tasks.

Moreover, in the preferred embodiment of the present invention the log file and the executable file are hidden files, making them invisible to the casual user.

The time logging program ceases operating after it has counted a predetermined number of timer interrupts. In addition, the time logging program counts the number of times the computer executes the TSR program, which generally corresponds to the number of times the computer is booted, and likewise ceases operating after it has counted a predetermined number of boots.

The time logging program can be activated with an input parameter to disable further operation.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is perspective view of a computer system, into which the present invention may be incorporated.

FIG. 2 is a simplified block diagram of a computer system, into which the present invention may be incorporated.

FIGS. 3A and 3B are flowcharts describing the operation of the preferred embodiment's Initialize function, FIG. 3 shows the relationship between FIGS. 3A and 3B.

FIG. 4 is a flowchart describing the operation of the preferred embodiment's DosTimer function.

FIG. 5 is a flowchart describing the operation of the preferred embodiment's DosIdle function.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. 1 illustrates an exemplary computer system **10** into which the present invention is advantageously incorporated. The computer system **10** comprises a base unit **11**, a monitor **12**, a keyboard **14**, and a flexible disk drive **16**. The keyboard **12** provides user access to the computer system **10**, and the monitor **14** displays data and other information generated by the computer system **10**. The flexible disk drive **16** permits a user to transfer software and data to and from the computer system **10**. The computer system **10** may advantageously include other peripheral devices, such as, for example, a printer (not shown).

FIG. 2 is a highly simplified block diagram of the computer system **10** showing a microprocessor or central processing unit (CPU) **50**, a random access read/write memory (RAM) subsystem **52**, and an input/output (I/O) subsystem **54**, connected to a common system bus **56**. A number of peripheral devices, such as, for example, a hard disk drive **60**, the floppy disk drive **16**, a timer subsystem **64**, the keyboard **14** and the monitor **12**, communicate with the microprocessor **50** via the I/O subsystem **54** and the system bus **56**. In addition, various components connected to the I/O subsystem **54** generate interrupts which are communicated to the microprocessor **50** via one or more interrupt lines **70**. One skilled in the art will appreciate that the I/O subsystem **54** may include a number of different buses, such as, for example, a peripheral component interconnect (PCI) bus, an Industry Standard Architecture (ISA) bus, a video local bus, and the like, which are not shown in FIG. 2. The computer system **10** is configured in a conventional manner and is not altered in any manner in order to incorporate the present invention other than to load and run the program described herein. Thus, it is not believed to be necessary to further describe the basic structure or operation of the computer system **10**.

The preferred embodiment of this invention is a Terminate and Stay Resident (TSR) computer program called "Tracker". The Tracker program is loaded automatically by the computer's DOS operating system whenever the computer is booted. Because the Tracker program is a TSR, it can operate while other computer programs are running. Moreover, the Tracker program's operation is invisible to the user, so that the user will not be alerted to its presence or tempted to disable it.

Briefly described, the Tracker program monitors the computer system's timer and, at predetermined intervals (by default, every thirty minutes), stores the amount of time intervals the computer system has been operating in a log file located on the computer's hard drive. The Tracker program also logs the number of times the computer system has been booted. After a sufficient period of time has been monitored, or after a sufficient number of boots have been logged, the Tracker program disables its logging function and furthermore prevents itself from running as a logger when the

computer is booted in the future. In an alternative embodiment, the Tracker program also stores the time and date when the computer system is first booted and the log file is created.

The Tracker program has three main components: (1) an Initialize function, which is called when the program is first loaded, and which sets up the program's counters and prepares the program to terminate and stay resident; (2) a DosTimer function, which intercepts and counts the number of DOS timer interrupts issued by the computer system's timer; and (3) a DosIdle function, which intercepts the DOS idle interrupt and, when enough timer ticks to reach one interval have been counted, updates a log file, located on the computer's hard disk, to reflect the total time the computer has been operating.

The Tracker program is designed to accept one (optional) command-line argument. This argument can be either a number representing a valid interval time in minutes, or one of the commands DELETE, LOCK, RESET, OR SHOW. (The effect of these commands will be described below.) If no argument is provided or if a valid interval time is provided, the Tracker program terminates but stays resident, and begins logging the time the computer is on. (If no argument is given, the default interval time of thirty minutes is used.) If, on the other hand, a command is given, the Tracker program performs the requested function and terminates, but does not stay resident.

In the preferred embodiment, the computer system **10** is an IBM-PC compatible computer using the DOS operating system. As is well known to most DOS users, when such a computer is turned on or restarted ("booted"), the DOS operating system executes a batch command file called "AUTOEXEC.BAT", if such a file exists. The AUTOEXEC.BAT file typically performs basic startup functions such as loading device drivers and setting up the DOS environment. In the preferred embodiment, the computer seller configures the AUTOEXEC.BAT file so that it executes the Tracker program (which is located on the computer's hard disk) without a command-line argument. In this way, the Tracker program loads and starts running, using the default interval time, each time the computer is booted. Alternatively, the computer seller may configure the AUTOEXEC.BAT file so that it executes the Tracker program using an interval time other than the default interval time.

For the reader's aid in following the description below, the Tracker program uses the following variables:

BootCount: the number of boots counted.

IntervalCount: the number of intervals counted.

ManualLock: a flag that, when set, stops the Tracker program from operating.

AutoLock: another flag that, when set, stops the Tracker program from operating.

Process: another flag that, when set, stops the Tracker program from operating.

DoUpdateFile: a flag that, when set, causes the Tracker program to update the log file.

IntervalMinutes: the number of minutes per interval.

IntervalTicks: the number of timer ticks per interval.

TicksLeft: the number of timer ticks remaining before IntervalTicks ticks have been counted.

MaxBoots: the maximum number of boots to count.

MaxIntervals: the maximum number of intervals to count.

FIGS. 3A and 3B illustrate the Tracker program's Initialize function, which is called as soon as the Tracker program has been loaded into memory and executed. Beginning at a step **100** in FIG. 3A, the Initialize function checks whether any command-line parameters were specified. If so, the parameter is tested at a step **102** to determine if it is a number representing an interval time, in minutes. In the preferred embodiment, the interval time is in the range 30 to 60 minutes, although other ranges may be used. An interval of 30 minutes causes the computer system to be activated only twice per hour to update the log file. Thus, in a system where the disk drive is powered down during periods of inactivity by the user, the disk drive is powered up infrequently to update the log file such that the user is not as likely to notice the disk activity than if the log file is updated more often.

On the other hand, interval times longer than 60 minutes significantly affect the Tracker program's accuracy in logging operating time. This is because whenever the computer is shut down or rebooted, the Tracker program will not record the fraction of an interval that it has already counted. Thus, the Tracker program consistently underestimates the amount of time the computer has been used, by an amount proportional to the interval time. Longer interval times, therefore, lead to larger underestimations.

If no argument is specified, the Initialize function sets IntervalMinutes to the default interval time of 30 minutes at a step **104**. If a valid interval was specified, the Initialize function sets IntervalMinutes to the specified interval time at a step **106**. Next, at a step **108**, the Initialize function tests whether it is possible for the Tracker program to terminate and stay resident (TSR). The Initialize function performs this test in a manner well known in the art by using the DOS multiplex interrupt (2F hex) to determine whether Microsoft® Windows™ is executing and whether the Tracker program is already resident. If the Initialize function determines that the Tracker program cannot terminate and stay resident, the program terminates at a step **109** (FIG. 3B).

The Initialize function next tests at a step **110** whether the Tracker log file exists. In the preferred embodiment, the log file is identified by the name "~AST.LOG", and is stored as a hidden file located in the local hard drive's root directory (usually the "C:\" directory). The log file is hidden so that the user will not notice it and will not be tempted to alter it.

If the log file does not exist, the Initialize function creates the log file at a step **112**. The log file contains information for the following variables: BootCount, IntervalCount, IntervalMinutes, ManualLock, and AutoLock. In an alternative embodiment, the log file also includes the date and time when the log file was first created when the consumer first booted the computer system. As discussed above, the log file is created so that it is a hidden file. When first created, the log file variables are all zeroed, except the IntervalMinutes variable, which is set to the current IntervalMinutes value (either the default of thirty minutes, or whatever interval time was specified in the command-line argument). The Initialize function further checks at a step **114** whether its attempt at creating the log file was successful. If not, the program terminates at the step **109** (FIG. 3B).

If the log file does exist, the Initialize function loads the data from the log file into its variables. Initialize then ensures at a step **116** that the log file's IntervalMinutes value is the same as the specified (or default) IntervalMinutes value. This check is necessary because the Tracker program logs the number of intervals counted, not the number of minutes counted, and therefore if the Tracker program is directed to log at intervals different from the log file's interval time, the IntervalCount value will have an indeterminate meaning. For this reason, if the log file's Interval-

Minutes value does not agree with the current IntervalMinutes value, the program terminates at the step **109** (FIG. 3B).

If the log file does exist and if the log file's IntervalMinutes value agrees with the current IntervalMinutes value, the Initialize function also ensures at a step **118** that neither its ManualLock flag nor its AutoLock flag is set. Either of these flags directs the Tracker program to stop operating. Therefore if either flag is set, the program terminates at the step **109** (FIG. 3B).

If the program proceeds through the foregoing tests without terminating, the Initialize function increments the BootCount variable at a step **120**, and then at a step **122** checks whether a predetermined maximum number of boots (MaxBoots) have been counted. In the preferred embodiment, MaxBoots is set to 1,500, representing the assumption that after 1,500 boots, the system is sufficiently used that further usage tracking is unnecessary. If MaxBoots (e.g., 1,500) boots have been counted, the Initialize function sets the AutoLock flag at a step **124**.

In any case, the Initialize function then updates the log file at a step **126** so that it contains the current set of logged variables. At this time, the Initialize function also calculates the number of ticks required per interval (IntervalTicks), and sets the TicksLeft variable to this number. IntervalTicks is equal to IntervalMinutes multiplied by 1,091, which closely approximates the 1,090.9 ticks per minute for an IBM-compatible computer's system timer when the computer system is executing under DOS. The Initialize function also calculates MaxIntervals, which sets the maximum number of intervals the Tracker program will count before disabling itself. In the preferred embodiment, MaxIntervals is equal to 90,000 minutes (1,500 hours) divided by IntervalMinutes, so that the Tracker program will log time for the first 1,500 hours the computer is on, and then stop.

Finally, at a step **128** the Initialize function chains the Tracker program into the DOS timer, idle, and multiplex interrupt vectors, and then terminates and stays resident at a step **130**. The process for hooking a TSR program into interrupt vectors is well-known in the art and need not be described here. It is necessary only to note that the Tracker program's DosIdle function is chained into the DOS idle (28 hex) interrupt vector, and that the Tracker program's DosTimer function is chained into the DOS timer (08 hex) interrupt vector, so that the DosIdle function is called whenever the DOS system is idle, and the DosTimer function is called at each DOS timer tick.

As illustrated in FIG. 3B, if the command-line parameter passed to the Tracker program was not a valid interval, the Initialize function checks at a step **200** whether the parameter is a valid keyword. As discussed above, valid keywords are DELETE, LOCK, RESET, and SHOW. If the parameter is not a valid keyword, the program terminates at the step **109**.

If on the other hand the parameter is a valid keyword, the Initialize function displays a brief heading giving the Tracker program name and copyright information, and then determines whether the log file exists at steps **202** and **204**. If no log file exists, the Initialize function displays an error message at a step **206** and then the program terminates at the step **109**.

If the log file exists at the step **204**, then the Initialize function responds to the keyword command in the command line as follows.

If the DELETE keyword is found in the command line at a step **210**, the Initialize function deletes the log file at a step **212** and then terminates at the step **109**.

If the LOCK keyword is found in the command line at a step **220**, the Initialize function sets the ManualLock flag and updates the log file at a step **222**. Then, at a step **224**, the Initialize function displays the logged data. The Initialize function then terminates at the step **109**.

If the RESET keyword is found in the command line at a step **230**, the Initialize function zeroes the BootCount, IntervalCount, ManualLock, and, AutoLock variables and updates the log file at a step **232**. The Initialize function then displays the logged data at the step **224** and then terminates at the step **109**.

If the SHOW keyword is found in the command line at a step **240**, then, at the step **224**, the Initialize function displays the number of hours logged (IntervalCount times IntervalMinutes divided by 60), the number of boots counted, the IntervalMinutes period, and whether the ManualLock or AutoLock flags are set. The Initialize function then terminates at the step **109**.

These keyword commands are not published as part of the system documentation, and are intended to be used only by the seller prior to shipping a new system or when a system is returned.

If the Tracker program has succeeded in terminating and staying resident, the Tracker program's DosTimer and DosIdle functions will be called whenever a system timer interrupt or DOS idle interrupt occurs, respectively.

The DosTimer function counts the number of timer ticks issued by the system timer. As illustrated in FIG. **4**, the DosTimer function's first operation is to check the Process flag at a step **300** to determine if the Tracker program is still intended to operate. This check is made early to ensure that, if the Tracker program should cease operation (if, for example, the requisite 1,500 hours logged have been reached), the DosTimer function exits quickly. Clearly, since the DosTimer function is called 1091 times each minute, it is advantageous for the function to quit as soon as possible if its continued operation is unnecessary. If the Process flag is not set, the DosTimer function returns control to the interrupt calling function at a step **302**.

If the Process flag is set, the DosTimer function then checks an internal flag (DosTimerExecuting) at a step **304** to determine if the DosTimer function has already been called. Although it is an unlikely event, it is possible for the DosTimer function to be called while the function is executing. This can occur if the system timer issues an interrupt during the DosTimer function's execution. For this reason, the DosTimer function must ensure that it is not already active, before continuing to execute. If the DosTimerExecuting flag is already set, the DosTimer function returns control to the interrupt calling function at the step **302**.

If the DosTimerExecuting flag is not set, the DosTimer function then decrements the TicksLeft variable at a step **306**, and then checks whether it has reached zero yet at a step **308**. Because TicksLeft was initialized to the value of IntervalTicks, when TicksLeft reaches zero, one interval has elapsed. If the TicksLeft variable is not zero, the DosTimer function returns control to the interrupt calling function at the step **302**. If TicksLeft is zero, the DosTimer function sets the DosTimerExecuting flag at a step **310**, resets TicksLeft to IntervalTicks at a step **312**, increments IntervalCount at a step **314**, and then sets the DoUpdateFile flag at a step **316**.

Next, the DosTimer function checks whether IntervalCount has reached MaxIntervals at a step **318**. If it has, the DosTimer function sets the AutoLock flag at a step **320**, which will (as discussed below) stop the Tracker program from operating further, after the log file has been updated one last time. The DosTimer function then clears the Dos-

TimerExecuting flag at a step **322** and returns control to the interrupt calling function at the step **302**.

If at the step **318**, the IntervalCount has not reached MaxIntervals, the DosTimer function clears the DosTimer-Executing flag at the step **322** and returns control to the interrupt calling function at the step **302**.

FIG. **5** illustrates the DosIdle function. The DosIdle function is called in response to a DOS idle interrupt, which is issued whenever DOS is idle. Like the DosTimer function, the DosIdle function first checks an internal flag (DosIdleExecuting) at a step **400** to ensure that it is not already executing. If it is already executing, the DosIdle functions returns control to the interrupt calling function at a step **402**. If it is not already executing, the DosIdle function checks whether the DoUpdateFile flag is set at a step **404**. Because the DosIdle function's only purpose is to update the log file, it need only operate when DoUpdateFile has been set.

If the DoUpdateFile flag is set, the DosIdle function prepares for operation by setting the DosIdleExecuting flag at a step **406**, and then checks the DOS system at a step **408** to ensure that it is ready to accept a write request to the system's local hard drive. As those with skill in the art will appreciate, this check is performed by checking the DOS CriticalError flag (whose location is determined at initialization time via a DOS interrupt (21 hex) function call) to ensure that no critical error has occurred in the DOS system.

If a DOS critical error would prevent the log file writing operation, the DosIdle function clears the DosIdleExecuting flag at a step **410** and then terminates by returning control to the interrupt calling function at the step **402**. If not, the DosIdle function updates the log file, by writing to it the BootCount, IntervalCount, IntervalMinutes, ManualLock, and AutoLock values at a step **412**. The DosIdle function then clears the DoUpdateFile flag at a step **414**.

The DosIdle function next checks whether either the ManualLock or AutoLock flag is set at a step **416**. If neither flag is set, the DosIdle function clears the DosIdleExecuting flag at a step **410** and returns at the step **402**. If either flag is set, the DosIdle function clears the Process flag (which will prevent the DosTimer function from further counting timer ticks) at a step **418**, clears the DosIdleExecuting flag at the step **410**, and returns control to the interrupt calling function at the step **402**.

The Tracker program thus provides a simple way of keeping track of the number of boots and the total operating time of the computer system **10** in a manner generally transparent to the system's user. Thus, if the computer system **10** is returned to the seller, the seller can readily determine, by examining the log file (i.e., by executing the Tracker program with the SHOW parameter in the command line), whether the computer system **10** has been used, and approximately how much it has been used.

This invention may be embodied in other specific forms without departing from the essential characteristics as described herein. The embodiments described above are to be considered in all respects as illustrative only and not restrictive in any manner. The scope of the invention is indicated by the following claims rather than by the foregoing description. Any and all changes which come within the meaning and range of equivalency of the claims are to be considered within their scope.

What is claimed is:

1. A system for tracking the amount of time a computer has been operated after delivering the computer to an end user, said system comprising:

a random access memory which stores data and executable programs while said computer is operating;

a source of periodic interrupts to said computer;

a non-volatile data storage device;

a log file located on said non-volatile data storage device, said log file containing data representing the mount of time said computer has been on; and

a time logging program which is loaded into memory and which is activated in background mode each time said computer is booted, said time logging program counting said timer interrupts and updating said data in said log fie when a predetermined number of said timer interrupts has been counted, said log file thereby storing a count representing a time duration for which said computer is operated, and wherein said time logging program ceases operating after it has counted a predetermined number of timer interrupts.

2. A system for tracking the amount of time a computer has been operated after delivering the computer to an end user, said system comprising:

a random access memory which stores data and executable programs while said computer is operating;

a source of periodic interrupts to said computer;

a non-volatile data storage device;

a log file located on said non-volatile data storage device, said log file containing data representing the amount of time said computer has been on; and

a time logging program which is loaded into memory and which is activated in background mode each time said computer is booted, said time logging program counting said timer interrupts and updating said data in said log file when a predetermined number of said timer interrupts has been counted, said log file thereby storing a count representing a time duration for which said computer is operated, and wherein said time logging program also counts the number of times said computer system is booted.

3. A system for tracking the amount of time a computer, which has a memory and a non-volatile data storage device and which issues timer interrupts, is on, comprising:

a log file located on said non-volatile data storage device, said log file containing data representing the amount of time said computer has been on; and

a background executing program which is loaded into said memory and activated each time said computer is booted, said background executing program intercepting and counting said timer interrupts, said background executing program updating said log file's data when a predetermined number of said timer interrupts has been counted, and wherein said time logging program ceases operating after it has counted a predetermined number of timer interrupts.

4. A system for tracking the amount of time a computer, which has a memory and a non-volatile data storage device and which issues timer interrupts, is on, comprising:

a log file located on said non-volatile data storage device, said log file containing data representing the amount of time said computer has been on; and

a background executing program which is loaded into said memory and activated each time said computer is booted, said background executing program intercepting and counting said timer interrupts, said background executing program updating said log file's data when a predetermined number of said timer interrupts has been counted, and wherein said time logging program also counts the number of times said computer system is booted.

* * * * *

US006119162A

United States Patent [19]

Li et al.

[11] Patent Number: 6,119,162

[45] Date of Patent: Sep. 12, 2000

[54] **METHODS AND APPARATUS FOR DYNAMIC INTERNET SERVER SELECTION**

[75] Inventors: **Chuang Li**, Saratoga; **Alex Weihwang Jeng**, San Jose; **Victor Maravillas**, San Francisco; **Chee Hin Ho**, San Jose, all of Calif.

[73] Assignee: **Actiontec Electronics, Inc.**, Sunnyvale, Calif.

[21] Appl. No.: **09/161,609**

[22] Filed: **Sep. 25, 1998**

[51] **Int. Cl.**[7] **G06F 15/16**

[52] **U.S. Cl.** **709/227**; 709/220; 709/221; 709/249

[58] **Field of Search** 709/220, 249, 709/250, 238, 227, 221, 228; 714/4

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,784,555	7/1998	Stone	709/220
5,835,481	11/1998	Akyol et al.	370/216
5,852,722	12/1998	Hamilton	709/221
5,862,348	1/1999	Pedersen	709/229
5,915,217	6/1999	Wiedeman et al.	455/427

OTHER PUBLICATIONS

Srisuresh et al, "Load Sharing using IP Networking Address Translator" Network Working Group, RFC 2391, pp 1–18, Aug. 1998.

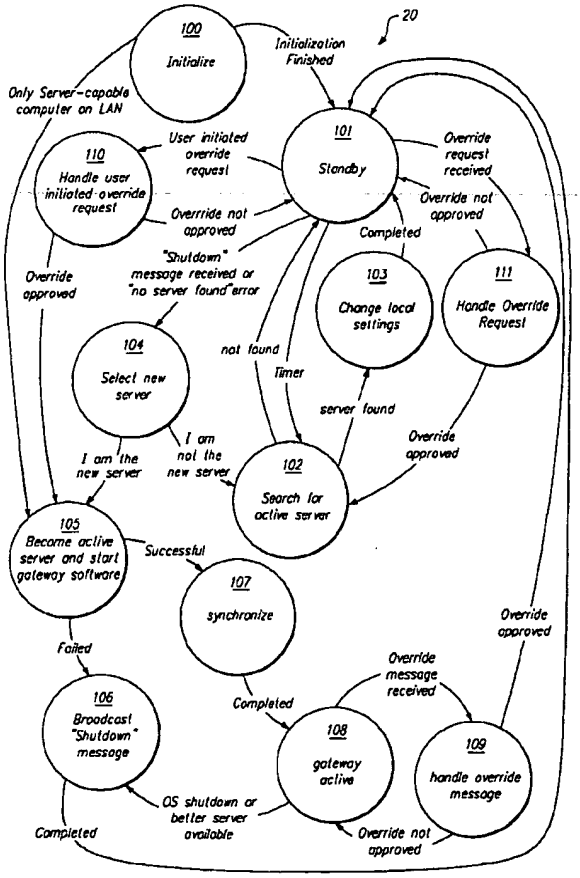
Primary Examiner—Mehmet B. Geckil

Attorney, Agent, or Firm—Fish & Neave; Nicola A. Pisano; Christopher J. Frerking

[57] **ABSTRACT**

Methods and apparatus for dynamically selecting a computer on a local area network to become a server providing shared access to a wide area network, such as the Internet, to all of the computers on the local area network is disclosed. A server may be selected from among any of the computers on the local area network that is currently available (i.e. powered on), and capable of establishing a connection to the wide area network. If the current server shuts down, a new server may be selected and started with only minimal impact on the other computers on the local area network, which are automatically reconfigured to use the new server to route network traffic to the wide area network.

39 Claims, 3 Drawing Sheets



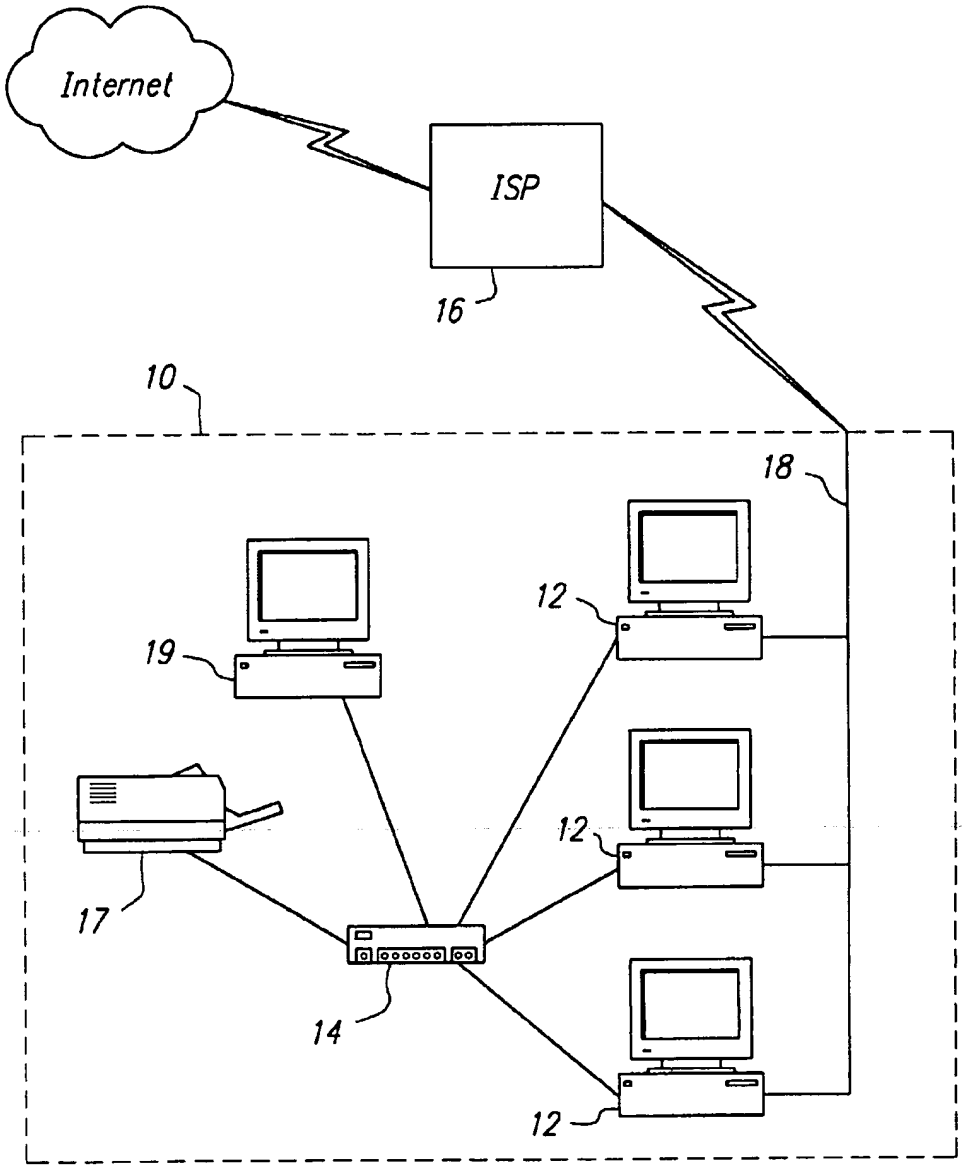


FIG. 1

FIG. 2

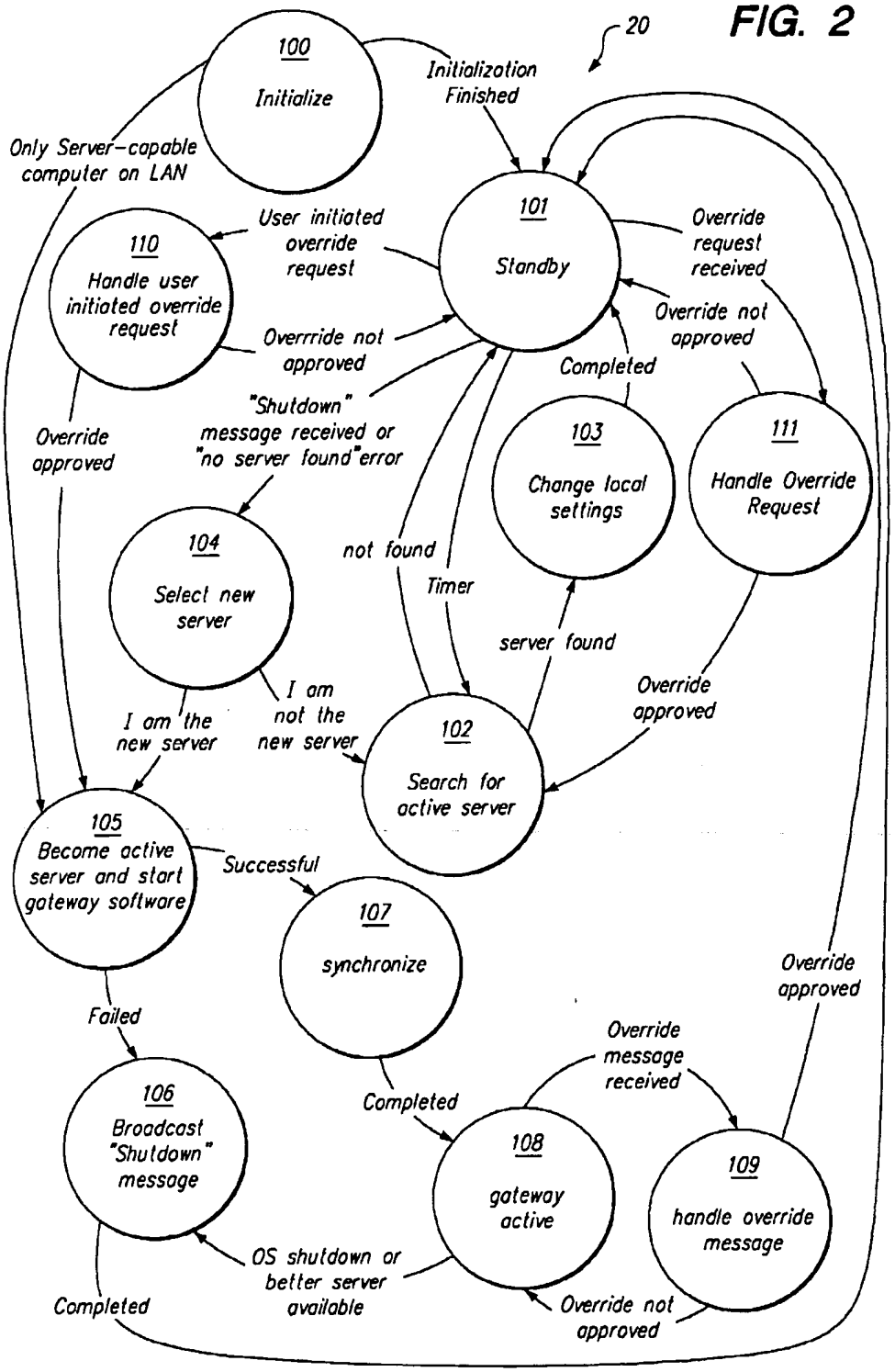
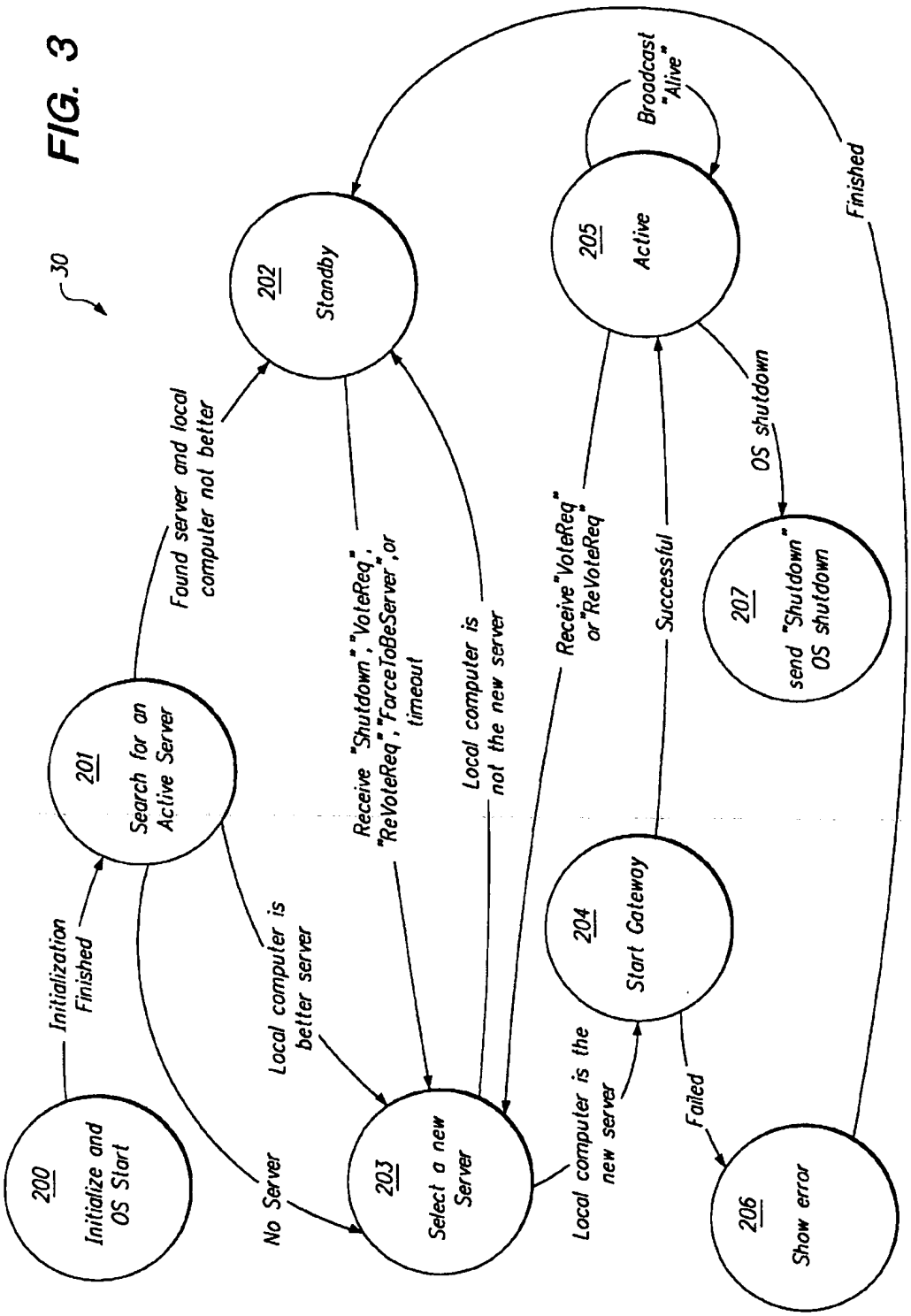

20
100
Initialize
Initialization Finished
Only Server-capable computer on LAN
101
Standby
110
Handle user initiated override request
User initiated override request
Override request received
Override not approved
Overrride not approved
Completed
"Shutdown" message received or "no server found" error
Override approved
103
Change local settings
111
Handle Override Request
104
Select new server
not found
Timer
server found
Override approved
I am not the new server
I am the new server
102
Search for active server
105
Become active server and start gateway software
Successful
107
synchronize
Override approved
Failed
Override message received
106
Broadcast "Shutdown" message
Completed
108
gateway active
109
handle override message
OS shutdown or better server available
Completed
Override not approved

FIG. 3


30
200 Initialize and OS Start
201 Search for an Active Server
202 Standby
203 Select a new Server
204 Start Gateway
205 Active
206 Show error
207 send "Shutdown" OS shutdown
Initialization Finished
Found server and local computer not better
Local computer is better server
No Server
Receive "Shutdown", "VoteReq", "ReVoteReq", "ForceToBeServer", or timeout
Local computer is not the new server
Receive "VoteReq" or "ReVoteReq"
Local computer is the new server
Successful
Failed
Broadcast "Alive"
OS shutdown
Finished

METHODS AND APPARATUS FOR DYNAMIC INTERNET SERVER SELECTION

FIELD OF THE INVENTION

The present invention relates generally to computer networking, and to establishing a connection to a wide area network, such as the Internet, that is shared by a small number of computers in a home environment.

BACKGROUND OF THE INVENTION

The explosive growth of the Internet that has occurred over the past few years has made the Internet and the World Wide Web (WWW) an increasingly important means for communication and distribution of information. Although much of the growth of the Internet has occurred due to its uses in education, research, and business, many households are now purchasing computer equipment and establishing Internet connections. As increasing numbers of households and families gain access to the Internet, numerous services, such as entertainment and shopping, are becoming available to serve the needs of these users. Home users are already becoming the predominant population of Internet users, and will further increase in numbers as the services available to home users become more numerous and attractive.

At the same time that increasing numbers of home users are gaining access to the internet, the price of computer equipment is rapidly decreasing. During the past year, personal computer systems priced at under $1000 were the fastest growing segment of the personal computer market. Almost all of these inexpensive computers come with a pre-installed high-speed modem and software for connecting to the Internet. With the availability of such inexpensive computers, it is not uncommon for households to have multiple computer systems, each capable of establishing a connection to the Internet.

The equipment necessary to connect multiple computer systems together to form a local area network (LAN) has also become inexpensive and simple. Many home computers are now equipped by the manufacturer with a standard network interface. Low cost add-on network interface cards and network hubs are also readily available. Setting up a LAN in the home is an attractive option for households having multiple computer systems, as it permits many resources, such as printers or storage space to be shared between all of the computers on the LAN. Moreover, manufacturers of other types of home equipment, such as home security systems, home control systems, audio and video equipment, and appliances are starting to incorporate network interfaces into their products. Already, many new homes are being built with the wiring for a LAN built-in, and it is expected that over the next decade, many more households will install a LAN.

Additionally, with the potential growth of home LANs, numerous networking technologies have been developed to make it easier to install a LAN in a home environment. These include technologies that can connect a LAN through preexisting wiring in a home, for example, by sending LAN traffic across power lines or home telephone lines. Also, numerous wireless LAN technologies that may be appropriate for home use have been developed, such as infrared and low-power RF LANs. As this type of equipment becomes widely available at a relatively low cost, it is expected that household LANs will become commonplace.

Gaining access to the Internet through a single computer is relatively simple. A large number of Internet service providers (ISPs) provide dial-up accounts that permit virtually unlimited low-speed access to the Internet for a modest monthly fee. All that is typically needed to connect to the Internet through an ISP is a personal computer equipped with a modem, a telephone line, and software (that typically is pre-installed on the computer system) for accessing the Internet through the ISP.

The speed of the modem typically determines the speed of the connection to the Internet, and is currently less than 56,000 bits per second. Over the next few years, various types of higher speed connections, such as cable modems or digital subscriber lines, capable of transferring more than a million bits per second, are expected to become widely available for home use at a relatively low cost.

In the past, dial-up connections typically provided access to the Internet to only a single computer at a time. Connecting multiple computers to the Internet required multiple telephone lines, and multiple ISP accounts. The monthly costs of maintaining multiple telephone lines and ISP accounts made this option prohibitively expensive for most households. As a result, even if a household had multiple computers, each with a high-speed modem, only one of these would typically be connected to the Internet through an ISP at any given time.

Recently, it has become possible to share a single dial-up connection to the Internet with all the computers connected to a LAN by using "gateway software." The two widely available types of gateway software are called "proxy server" software, and "network address translation" software. Proxy server software works by providing an intermediary network server between the Internet and the LAN. Computers on the LAN are configured to send their requests to the proxy server software running on one particular computer on the network. The proxy server software then sends the request to the appropriate place on the Internet, receives any response, and sends the response back to the appropriate computer on the LAN. Thus, the proxy server interposes itself in every communication between a computer on the LAN and the Internet.

Network address translation software works in a manner similar to proxy server software, but is somewhat more transparent to the other computers on the LAN. Network traffic addressed to computers outside of the LAN is directed to the network address translation software, which reroutes the network traffic to the Internet. The addresses of network traffic received from the Internet in response to requests made by computers on the LAN are translated to reroute the traffic to the appropriate computers on the LAN.

Both proxy server software and network address translation software are commercially available from numerous vendors, and may be run on a wide variety of platforms. One popular proxy server software package for use with Microsoft's WINDOWS 95 operating system is WINGATE, produced by Deerfield.com, of Gaylord, Mich. A popular network address translation software package for use with Microsoft's WINDOWS 95 operating system is SYGATE, produced by SyberGen Incorporated, of Fremont, Cali.

Using such software, one of the computers on the network (hereinafter referred to as the "server") establishes a connection to the Internet (e.g. using a modem and ISP), and permits the other computers on the LAN to access the Internet through the server. Typically, only one of the machines on the LAN is designated as the server, and only the server may normally establish a connection to the Internet.

One drawback of this arrangement is that the software typically requires that one of the computers be designated as

a server, and the server may not be easily changed once designated. Since all of the computers on the LAN rely on the server for their internet connection, if the server is down (e.g. powered off or temporarily disconnected from the LAN), none of the computers on the LAN are able to access the Internet. Similarly, if the server stops functioning (e.g. due to a software problem or a system crash) while other computers on the LAN are using the server to access the Internet, all of the Internet connections are immediately lost.

These difficulties are easily overcome in a small business environment, where it is common to dedicate a specific computer to performing the tasks of a server. The server is typically not used for any other functions (aside from sharing of other network resources), and runs continuously. The server is also typically configured with a professional, stable operating system, and may be configured to run only a small set of carefully selected software packages that are known to be stable, and are unlikely to encounter incompatibilities or to cause system crashes. It is also not uncommon for small businesses to hire a computer professional to make certain that their server is properly configured, and continues to function.

Conditions typically are very different in a home environment. For example, even if the household owns several computers, rarely is one of them dedicated to being a server. The computers used in the home are often turned off during non-use, so there is no guarantee that any particular computer will be powered on and thus available for use as a server. Additionally, many home users run somewhat unstable consumer-level operating systems on their computers, and also may run a wide variety of software, increasing the probability of incompatibilities, system crashes, and other software-related problems. It is difficult to use typical gateway software in such an environment, since no one computer on the LAN can be reliably designated as the server.

In view of the above, it would be desirable to provide a means for dynamically selecting which of a number of available computers on a LAN should be used as a server to provide LAN access to the Internet.

It would also be desirable to provide a means for dynamically and transparently switching between servers when the computer that is currently acting as server is shut down.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a means for dynamically selecting which of a number of available computers on a LAN should be used as a server to provide LAN access to the Internet.

It is also an object of the invention to provide a means for dynamically and transparently switching between servers when the computer that is currently acting as server is shut down.

These and other objects of the present invention are achieved by providing "server selection agent" software that works with gateway software to dynamically select and switch between servers. The software is designed to reduce the difficulties encountered when running a server in a home environment by permitting any one of a number of modem-equipped computers on a LAN (i.e. most home computers) to become a server.

The software first searches the LAN to see if there is already an active server. If no server is found, then the software selects which of the available computers on the LAN should become the server. Once the selection is made, the computer that was selected as the server starts the gateway software, and the other computers route their Internet traffic through the selected server. When the selected server shuts down, the computers on the LAN choose a new server from among the available computers, and resynchronize their network traffic to use the new server.

Except in certain special override conditions, this entire process may be accomplished in a manner that is transparent to the users of the computers. Users of the computers on the LAN simply run their Internet-capable software as if they had a permanent connection to the Internet. The server selection agent software and gateway software run in the background, and automatically handle all of the details of selecting a server, connecting to an ISP to gain access to the Internet, routing network traffic between the LAN and the Internet, and dynamically switching between servers.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and other objects and advantages of the present invention will be apparent upon consideration of the following detailed description, taken in conjunction with the accompanying drawings, in which like reference characters refer to like parts throughout, and in which:

FIG. **1** shows a LAN on which the methods of the current invention may be used;

FIG. **2** shows a state diagram of a preferred embodiment of the present invention; and

FIG. **3** shows a state diagram of an alternative preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

Referring to FIG. **1**, typical home or small office LAN **10** is described. Each of computers **12** on LAN **10** includes a network interface card and a modem. Computers **12** are coupled to LAN **10** through their network interface cards, and through network hub **14**, which interconnects all of the devices on LAN **10**.

Each of computers **12** has an associated modem (e.g. an internal modem) or other communications device through which the computer can connect to a wide area network. For purposes of this application a wide area network (WAN) comprises any network or communications system outside of the LAN. It also will be evident to one skilled in the art that a connection to a wide area network may be established either directly, or through an intermediary network (LAN or WAN).

Typically, computers **12** will establish a connection to the Internet (or other WAN) by connecting to ISP **16** through public telephone line **18**. Because LAN **10** is being used in a home or small office, all of computers **12** share a single public telephone line **18**, so only one of computers **12** may use its modem to connect to ISP **16** (or any other service) at any given time. Other network capable devices, such as printer **17** may also be connected to LAN **10**. Additionally, computers without modems, such as computer **19**, may be connected to LAN **10**. Such computers may not be connected to public telephone line **18**, and may not establish a connection to ISP **16**.

In a typical previously known LAN, only one of computers **12** on LAN **10** could be connected to the Internet through ISP **16** at any given time. Whichever one of computers **12** was connected would block any of the other computers on the LAN from using public telephone line **18** and the Internet account on ISP **16**. The development of gateway software, however, such as network address translation

software or proxy server software, permits all of the computers on LAN **10** to share public telephone line **18** and access to ISP **16**. The gateway software usually is installed on one of computers **12**, which then functions as a designated server. All of the other computers **12** and computer **19** are configured to route any traffic to the Internet through the designated server containing the gateway software. Only the designated server normally uses its modem to connect to ISP **16**, and if the server is shut down or disconnected from LAN **10**, all of the other computers and devices on LAN **10** become unable to access the Internet.

In accordance with the principles of the present invention, server selection agent software is provided that enables any of computers **12** to be dynamically selected as the server. The server selection agent software of the present invention may be included as part of an operating system, or may be a separate software application. In either case, the server selection agent software is installed on all of computers **12** and on computer **19**. Additionally, gateway software is installed on all of computers **12**, since any of computers **12** could potentially become the server.

When any of computers **12** or computer **19** needs access to the Internet, a server is selected from among the available computers **12**. Computer **19** cannot be selected as a server, because it does not have a modem, and cannot establish a connection to ISP **16**.

If the server is shut down or disconnected from the network, the other computers select a new server from the available computers **12**, and reestablish their connections to the Internet through the new computer **12** selected to function as the server. In a preferred embodiment, the server may also be changed if a new one of computers **12** becomes available that would be a better server (e.g. because it has a faster modem) than the current server. Only if none of computers **12** are available, such as if they are all powered off, would other computers (e.g. computer **19**) or devices on LAN **10** be unable to access the Internet.

A preferred embodiment of the server selection agent software of the present invention also permits the user of one of the computers to request that his or her computer become the server. This may be useful, for example, if the user wishes to connect to a different ISP than the one that is currently in use by the server. If this "Override" mode is used, a message is displayed on each of the active computers on LAN **10**, asking permission to switch servers. If all the users of the active computers on LAN **10** consent by selecting an "OK" option in the message, the server is switched, and all of the computers on LAN **10** reroute their Internet traffic through the new server (i.e. they become "clients" of the new server).

It will be apparent to one skilled in the relevant arts that the modems in computers **12** may be replaced with cable modems, ISDN modems, or digital subscriber line modems, and public telephone line **18** may be replaced with other communications technology, such as a cable line, an ISDN line, or a digital subscriber lines. It will also be apparent that computers **12** may not all be identical. Some of computers **12** may be faster than others, or have faster modems. Some of computers **12** may be connected to ISP **16** through, for example, a cable line, while others of computers **12** are connected to ISP **16** through public telephone line **18**.

Other configurations having multiple public telephone lines (or other communications lines) may also be used. With multiple telephone lines, it is possible to have more than one active server, and each of the client computers may select one of the active servers to reroute its Internet traffic. Alternatively, each of the computers may connect to more than one active server, to provide additional network bandwidth.

Additionally, the "hub and spoke" configuration of LAN **10** is for illustration only. Other network configurations also are possible, and may not require use of network hub **14**. Additionally, although LAN **10** typically comprises standard 10Base-T or 100Base-T connections, it could also comprise connections made via power lines, telephone lines, wireless connections made via infrared or RF transmission, or any other type of network connection. The principles of the present invention may be applied equally to any of these configurations, and should not be seen as limited to the configuration of LAN **10** shown in FIG. **1**.

Referring now to FIG. **2**, a state diagram of a preferred embodiment of the server selection agent software of the present invention is shown. This software is installed on all of the computers on a LAN, either as a part of their operating systems, or as a separate application, and starts executing on a computer when the computer's operating system is started. Server selection agent software **20** normally executes in the background, and interferes only minimally with other software executing on the computer. A typical computer user may not even be aware that server selection agent software **20** is executing. Optionally, the software may display a small icon on the screen of the computer on which it is executing to provide an indication of the current status of the software. By executing a separate monitoring program, a user may be able to receive more detailed information on the status and functioning of server selection agent software **20**.

Execution of server selection agent software **20** starts in state **100**. In state **100**, a powered-on computer, hereinafter called the "local computer," performs initialization, and establishes its presence on the LAN. This involves steps of broadcasting a initialization message to all other computers on the LAN, the initialization message containing information on the capabilities of the computer, such as the modem speed of the associated modem, and the CPU speed. The local computer then waits a short time to receive messages from all of the other active computers on the LAN detailing the capabilities of those machines.

In a preferred embodiment, the information broadcast during the initialization step includes information on the speed of any modems or other communication devices that the local computer may use to establish a connection to an ISP, and information on the overall system speed of the computer. This information is typically gathered once, when the software is installed. Information on modems or other communications devices may be provided by the user during installation, or may be automatically determined. Information on the overall system speed of the local computer may be determined by executing a benchmark program designed to test how well the computer will perform as a server. This information may be updated whenever significant changes are made to the computer, such as when new hardware is installed.

If the local computer receives no messages from other computers on the LAN, or receives messages only from computers that cannot become a server (e.g. because they do not have a modem), it concludes that there are no other currently active computers capable of becoming a server. In this case, if the local computer is capable of becoming a server, it switches to state **105** and becomes the currently active server. If none of the active computers on the LAN, including the local computer, are capable of becoming a server, an error message is displayed, and the local computer

switches to state **101**. Under more normal conditions, when the local computer receives information on the capabilities of other currently active computers on the LAN, the local computer switches to state **101**.

State **101** is a standby mode, where the software waits for some event, such as a timer message or broadcast message from other computers on the LAN to cause it to transition to a different state. Since state **101** requires little or no processing, and software **20** spends most of its time in state **101**, the processing demands placed on a computer by software **20** are minimal.

In state **101**, when a timer event occurs, which preferably happens at one minute intervals, the local computer enters state **102**, which searches for an active server. If a "Shutdown" message is received, the local computer transitions to state **104**, where a new server is selected. The local computer may also transition to state **104** if a "no server found" error occurs, as a result of the local computer repeatedly searching for an active server without finding one. If the user initiates an override request, described in detail hereinafter, the local computer transitions to state **110**, to handle the override request. If the local computer receives an override request broadcast in state **101**, it transitions to state **111**, to determine whether the override request is approved.

State **102** searches the LAN for an active server. This may be achieved by the local computer broadcasting a network message to all computers on the LAN requesting that they respond if they are the active server. If none of the computers responds to the request within a reasonable time (typically less than a second), then no server has been found, and the local computer displays an error message indicating that no active server could be found, and returns to state **101**. If another computer responds that it is the active server, the local computer transitions to state **103**.

It should be noted that the periodic searching for a server shown in FIG. **2** in the operation of states **101** and **102** could be changed without departing from the invention. For example, the active server could broadcast a message identifying itself at regular intervals, rather than having all of the computers on the LAN search for the server periodically.

In state **103**, the local computer adjusts or maintains local software settings to direct network traffic to the proper active server, and returns to state **101**.

In state **104**, a new server is selected. In a preferred embodiment, this involves the local computer examining the capabilities of all active computers on the LAN, and applying rules to determine which of the computers should become the server. Since all of the computers on the LAN apply the same criteria to determine which of them should become the server, and all of them have the same information on the capabilities of the computers on the LAN, all of the computers will make the same selection.

If it is determined that the local computer should become the server, then the local computer transitions to state **105** and becomes the active server. Otherwise, some other computer on the LAN becomes (or remains) the server, and the local computer transitions to state **102**, to search for the active server.

In one embodiment, the rules that determine which of the computers should become the server may be relatively simple. First, to become a server, a computer must have a modem or other communication device with which it can establish a connection to an ISP. If the local computer has no modem, it cannot be the active server. Next, the computer with the fastest modem should become the active server. If the information on capabilities indicates that any of the computers on the network has a faster modem than the local computer, then the local computer should not become the active server. Next, if there was a tie on modem speed, then the computer with the highest overall system speed should become the server. If the information on capabilities indicates that the modem speed of the local computer is the same as at least one other computer on the LAN, and the system speed of the local computer is less than another computer on the LAN having the same modem speed, then the local computer should not become the active server. Finally, in the event that there is a tie on both modem speed and overall system speed, then the computer with the lowest network address is selected as the server.

It will be evident that there are many other rules that could be applied to determine which computer should become the server. The determination could be made, for example, based a weighted sum of the information on the capabilities of each computer on the LAN. Alternatively, other capability information, such as average actual throughput when connected to the ISP could be used as a factor in deciding which computer should become the active server. Additionally, the selection process could be done by a single computer, such as the currently active server, rather than being done simultaneously on all of the active computers on the LAN.

In state **105**, the local computer becomes the active server by starting the gateway software. In a preferred embodiment, the gateway software comprises a modified version of a network address translation software package, such as SYGATE, by SyberGen Incorporated, of Fremont, Cali.

Alternatively, the gateway software may comprise proxy server software. Proxy server software is somewhat less preferred, because it is more difficult to configure. If proxy server software is used, then server selection agent software **20** should be modified to act as a local proxy server that runs on every computer on the LAN, and redirects messages to the "actual" proxy server software (i.e. a proxy cascade) running on the selected server, that in turn redirects network traffic to the Internet. This indirect approach avoids the need to reconfigure all of the Internet capable software on the computer every time the server changes.

In one embodiment, the gateway software is modified for use with server selection agent software **20** so that the gateway software and server selection agent software **20** can communicate with each other, and (optionally) so that the gateway software can shut down when an override request is processed, or when the rules indicate that a better server has become available. Alternatively, server selection agent software **20** may use standard operating system services to monitor unmodified gateway software, or to shut the gateway software down when necessary.

If the gateway software fails to start, the local computer transitions to state **106**, in which a "Shutdown" message is broadcast. The local computer then transitions back to state **101**. If the gateway software starts successfully, the local computer transitions to state **107**, which synchronizes network traffic between the new server and the other active computers on the LAN. The local computer then transitions into state **108**.

State **108** is a state in which server selection agent software **20** interacts with the gateway software. If a remote or local Internet request is received and the server is not already connected to the ISP, a connection is established. When notice is received that the operating system is shutting down (e.g. because the user is shutting down the computer), the local computer transitions to state **106**. If an override request is received, the local computer transitions to state **109**.

Additionally, if the local computer is in state **108**, and an initialization broadcast is received, the rules described above with respect to state **104** may be applied to determine if the newly initialized computer would be a better server. If so, the local computer may optionally transition to state **106**, shutting down as server, and allowing the newly initialized computer to become the active server.

In state **109**, the local computer (which is also the active server) handles receipt of an override request. A message is displayed on the screen asking the user of the local computer if he or she approves the override request. If all users of active computers on the LAN agree to the request, then the local computer shuts down the gateway software, stops acting as the server, and transitions back to state **101**. If any user on the LAN refuses to allow the override request, or a timeout occurs, then the local computer returns to state **108**, and remains the active server.

State **110** handles a user-initiated override request. When the user of the local computer requests that the local computer become the active server, approval is needed from all other active computers on the LAN. State **110** broadcasts an override request message, and waits to hear back from all of the active computers on the LAN, or for a timeout to occur. If all of the other active computers on the LAN approve the override request, then the local computer transitions to state **105** and becomes the active server. Otherwise, the local computer informs the user that the override request was unsuccessful, and returns to state **101**.

State **111** handles the receipt of an override request when the local computer is in state **101**. A message is displayed on the screen asking the user if he or she approves the override request. If the user approves of the request, the local computer broadcasts approval. If the user disapproves or a timeout occurs, the local computer broadcasts disapproval of the request. If all of the active computers on the LAN send approval, the request has been approved, and the local computer transitions to state **102**, to search for the new server. Otherwise, the local computer returns to state **101**.

Referring now to FIG. **3**, a state diagram of an alternative preferred embodiment of the server selection agent software of the present invention is described. Server selection agent software **30** may be more preferred than server selection agent software **20** of FIG. **2**, since it is somewhat simpler, and more robust handling errors. Server selection agent software **30** executes on every computer on a LAN, and may be either a part of the operating systems of the computers on the LAN or may be a separate application. Server selection agent software **30** starts executing in state **200** when the operating system is started.

State **200** is an initialization state, in which server selection agent software **30**, executing on a local computer, initializes itself and determines the capabilities of the active computers on the LAN. The information on the capabilities of the active computers on the LAN preferably comprises information on whether each computer has a modem or other communication device with which it can communicate with a wide area network, the speed of any modems or communication devices, and information on the overall system speed. This information may be gathered in a manner similar to that described hereinabove with reference to FIG. **2**. When initialization is finished, the local computer transitions to state **201**.

In state **201**, the local computer searches for an active server. This may be achieved by the local computer broadcasting a network message to all computers on the LAN requesting that they respond if they are the active server. If another computer on the LAN responds that it is the active server, then the local computer applies a set rules to determine if the local computer would be a better server than the current active server. The rules that are used to make this determination, for example, may be identical to the rules described with reference to state **104** of FIG. **2**.

If, based upon application of the foregoing rules, it is determined that the local computer would be a better server, then the local computer broadcasts a "VoteReq" message to all of the computers on the LAN, and transitions to state **203**. Otherwise, if the rules determine that the local computer is not better than the active server, then the local computer transitions to state **202**.

If no computer on the LAN responds that it is the active server, then the local computer transitions to state **203** to choose a server. Additionally, if an error condition occurs, such as if more than one computer responds that it is the active server on a LAN that can have only one active server, state **202** may force all of the systems on the LAN to choose a new server. This is achieved by broadcasting a "ReVoteReq" message to all of the computers on the LAN, and transitioning to state **203**.

State **202** is a standby state, in which server selection agent software **30** waits for some event, such as a broadcast message from other computers on the LAN, or a timeout condition, to cause it to transition to a different state. If a "Shutdown" message, a "ReVoteReq" message, a "VoteReq" message, or a "ForceToBeServer" message (described hereinafter) is received while the local computer is in state **202**, the local computer transitions to state **203** to select a new server.

Additionally, the local computer expects to receive an "Alive" message from the active server at regular intervals. If such a message is not received for a specified length of time, a timeout condition occurs. A timeout condition typically indicates that the active server is no longer functioning due to some problem, such as if the active server freezes or crashes. When a timeout condition occurs, the local computer transitions to state **203**, to select a new server. Optionally, the local computer may also broadcast a "ReVoteReq" message to all the active computers on the LAN, to let all the computers know that a new server should be selected.

It should be noted that the timeout condition of state **202** operates in a very different manner than the timer event described with reference to state **101** of FIG. **2**. The timer event causes a kind of "polling" to occur, in which each of the computers searches for the server at regular intervals. Using the timeout event of state **202**, the computers do not repeatedly search for a server. Instead, the active server periodically broadcasts "Alive" messages to let the computers on the LAN know that it is still functioning. It will be evident to one skilled in the art that the "polling" mode described with reference to FIG. **2** could be applied to the software shown in FIG. **3**, or the timeout mode described with reference to FIG. **3** could be applied to the software shown in FIG. **2**.

State **202** also handles user initiation of an override request. If the user of the local computer decides that he or she wants the local computer to become the server, an override request may be initiated from state **202**. If the user initiates an override request, the local computer broadcasts a "ForceToBeServer" message, and transitions to state **203**.

In state **203**, a new server is selected. If the local computer reaches state **203** because there was no active server on the network, due to a timeout, due to an error, or because a

"ReVoteReq" message was received, then the local computer applies a set of rules to determine which of the active computers on the LAN should become the server. The rules applied may be identical to the rules described hereinabove with reference to FIG. **2**. If the rules determine that the local computer should become the active server, then the local computer transitions to state **204**. Otherwise, if the rules determine that the local computer should not become the active server, the local computer transitions to state **202**. If no computer on the LAN can become the active server (e.g. because none of the active computers on the LAN has a modem), then an error message is displayed, and the local computer transitions to state **202**.

If the local computer reaches state **203** due to a "VoteReq" message, a "Shutdown" message, or a determination that the local computer would be a better server than the active server, then an additional step is required. For a vote to occur in these conditions, the consent of the users of other active computers on the LAN is needed. A message will be displayed on each of the computers on the LAN, requesting the user's consent. If consent is given by all of the computers on the LAN, then the process of selecting a server will proceed, as described hereinabove. If consent is not given, then the active server will remain active, and the other computers on the LAN will return to state **202**.

It should be noted that this is a different procedure than is used in the software described with reference to FIG. **2**, since permission is needed for the active server to shutdown, or for a computer that is better than the active server to become the active server. In the software described with reference to FIG. **2**, no permission from other users is needed to perform these functions. It will be evident to one skilled in the art that a permission scheme similar to the one described with reference to state **203** also could be applied to the software of FIG. **2**. Similarly, state **203** could be altered so that these functions are performed without permission of the users of other computers.

State **203** also handles override requests. If the local computer reaches state **203** due to a user initiated override request, or a "ForceToBeServer" message, then consent is needed before the computer that made the request may become the active server. Each of the active computers on the LAN displays a message requesting consent from the user for the override request. If consent is given by the users of all of the active computers on the LAN, then the computer that made the override request becomes the server, by transitioning to state **204**, while the other computers on the LAN transition to state **202**, and become clients of the new server. Otherwise, if consent is not given by all of the users of active computers on the LAN, then the active server remains active, and all of the other computers on the LAN transition to state **202**.

The local computer enters state **204** if it is decided through the server selection process that the local computer should become the active server. In state **204**, the gateway software is started. The gateway software preferably comprises modified gateway software, but may also comprise unmodified gateway software, as discussed hereinabove with reference to FIG. **2**. If the gateway software starts successfully, the local computer transitions to state **205**. Otherwise, if the gateway software fails to start, the local computer transitions to state **206**.

State **205** is a state in which server selection agent software **30** interacts with the gateway software. If a remote or local Internet request is received and the server is not already connected to the ISP, a connection is established. When notice is received that the operating system is shutting down (e.g. because the user is shutting down the computer), the local computer transitions to state **207**. If a "VoteReq" message or a "ReVoteReq" message is received, then the local computer transitions to state **203**. Additionally, the local computer (that is also the active server) periodically broadcasts "Alive" messages to all of the other computers on the LAN, to let the other computers know that the active server is still functioning.

In state **206**, an error message is displayed, informing the user of the local computer that the gateway software failed to start. The local computer then transitions to state **202**. Optionally, the local computer may remove itself from the set of computers that may become the active server prior to switching to state **202**. Additionally, the local computer may optionally broadcast a "ReVoteReq" message, to let the other computers on the LAN know that an error has occurred, and it is necessary to select a new server.

In state **207**, the local computer broadcasts a "Shutdown" message, and waits for the Operating system to shut down. It may be necessary for the local computer to refuse to shut down, and to return to state **205** if the users of the other active computers on the LAN refuse to give their consent for the local computer to stop being the active server.

Although preferred illustrative embodiments of the present invention are described above, it will be evident to one skilled in the art that various changes and modifications may be made without departing from the invention. For example, if there are multiple telephone lines, the server selection agent software may be altered to handle more than one active server, and the state that searches for active servers may determine which or the active servers to use based on the load at each active server. Additionally, if there are multiple active servers, each computer may route network traffic through more than one active server to increase network bandwidth.

Additionally, the mechanism for obtaining consent for an override, or for other functions that require consent could be changed so that a simple majority is needed, or so that only the approval of the active server, or of a network administrator is needed. As discussed above, the server selection rules may be altered to change the criteria used to determine which computer is selected as the active server.

It is intended in the appended claims to cover all such changes and modifications that fall within the true spirit and scope of the invention.

What is claimed is:

1. A method of sharing a connection to the Internet among a plurality of computers connected to a local area network, the method comprising:

(a) determining that a subset of the plurality of computers are available for use as a server;

(b) automatically selecting one of the subset of the plurality of computers to become the server;

(c) repeating steps (a) and (b) when the server becomes unavailable;

(d) establishing a connection between the server and the Internet; and

(e) routing Internet traffic through the server to the Internet.

2. The method of claim **1**, wherein routing Internet traffic through the server to the Internet comprises a step of running gateway software on the server.

3. The method of claim **2**, wherein the gateway software comprises network address translation software.

4. The method of claim 2, wherein the gateway software comprises proxy server software.

5. The method of claim 1, wherein determining that a subset of the plurality of computers are available for use as a server further comprises gathering information on selected capabilities of active ones of the plurality of computers.

6. The method of claim 5, wherein automatically selecting one of the subset of the plurality of computers to become the server comprises evaluating a set of rules to determine which of the subset should become the server.

7. The method of claim 6, wherein evaluating the set of rules comprises evaluating the information on selected capabilities of active ones of the plurality of computers.

8. The method of claim 7, wherein the selected capabilities include a communication speed of each of the active ones of the plurality of computers, and evaluating the set of rules comprises evaluating the communication speed.

9. The method of claim 7, wherein the selected capabilities include a system speed of each of the active ones of the plurality of computers, and evaluating the set of rules comprises evaluating the system speed.

10. The method of claim 6, wherein the method further comprises reevaluating the set of rules when an additional computer becomes active.

11. The method of claim 1, wherein determining that a subset of the plurality of computers are available for use as a server further comprises determining which of the plurality of computers are powered on.

12. The method of claim 1, wherein determining that a subset of the plurality of computers are available for use as a server further comprises determining which of the plurality of computers has access to a communication device capable of establishing a connection to the Internet.

13. The method of claim 1, wherein the method further comprises permitting a user of one of the subset of the plurality of computers to override automatic selection of the server.

14. The method of claim 13, wherein permitting a user to override automatic selection of the server further comprises overriding automatic selection of the server only if the users of all of the subset of the plurality of computers consent.

15. Apparatus for selecting a computer on a local area network to become a server to permit sharing of an Internet connection among a plurality of computers connected to a local area network, the apparatus comprising a local computer connected to the local area network, the local computer programmed to:

(a) determine that a subset of the plurality of computers are available for use as a server;

(b) automatically select one of the subset of the plurality of computers to become the server;

(c) repeat steps (a) and (b) when the server becomes unavailable;

(d) establish a connection between the local computer and the Internet, and become the server, if the local computer was selected to become the server; and

(e) route Internet traffic through the server to the Internet.

16. The apparatus of claim 15, wherein the local computer is further programmed to inform other active ones of the plurality of computers that the server is unavailable if the local computer is the server, and the local computer is becoming unavailable.

17. The apparatus of claim 15, wherein the local computer is further programmed to execute gateway software to route Internet traffic to the Internet, if the local computer is the server.

18. The apparatus of claim 17, wherein the gateway software comprises network address translation software.

19. The apparatus of claim 17, wherein the gateway software comprises proxy server software.

20. The apparatus of claim 17, wherein the local computer is further programmed to gather information on selected capabilities of active ones of the plurality of computers.

21. The apparatus of claim 20, wherein the local computer is programmed to automatically select one of the subset of the plurality of computers to become the server by evaluating a set of rules to determine which of the subset should become the server.

22. The apparatus of claim 21, wherein evaluating the rules comprises evaluating the information on selected capabilities of the active ones of the plurality of computers.

23. The apparatus of claim 22, wherein the selected capabilities include a communication speed of each of the active ones of the plurality of computers, and the evaluating the set of rules comprises evaluating the communication speed.

24. The apparatus of claim 22, wherein the selected capabilities include a system speed of each of the active ones of the plurality of computers, and evaluating the set of rules comprises evaluating the system speed.

25. The apparatus of claim 21, wherein the local computer is further programmed to reevaluate the set of rules when an additional computer becomes active.

26. The apparatus of claim 15, wherein the local computer is programmed to determine that a subset of the plurality of computers are available for use as a server by determining which of the plurality of computers are powered on and have access to a communication device capable of establishing a connection to the Internet.

27. The apparatus of claim 21, wherein the local computer is further programmed to permit a user of one of the subset of the plurality of computers to override automatic selection of the server.

28. The apparatus of claim 27, wherein the local computer is programmed to permit a user to override automatic selection of the server only if the users of all of the subset of the plurality of computers consent.

29. Apparatus for sharing an Internet connection among a plurality of computers, the apparatus comprising:

a local area network interconnecting the plurality of computers; and

a communication line permitting a connection between at least one of the plurality of computers and the Internet;

wherein each one of the plurality of computers comprises a processor programmed to:

(a) determine that a subset of the plurality of computers are available for use as a server;

(b) automatically select one of the subset of the plurality of computers to become the server;

(c) repeat steps (a) and (b) when the server becomes unavailable;

(d) establish a connection between the computer and the Internet, and become the server, if the computer was selected to become the server; and

(e) route network traffic through the server to the Internet.

30. The apparatus of claim 29, wherein the local area network comprises a network hub, and each of the plurality of computers is connected to the local area network through the network hub.

31. The apparatus of claim 29, wherein at least one of the plurality of computers comprises a communication device coupled to the communication line.

32. The apparatus of claim 31, wherein the communication device comprises a modem, and the communication line comprises a public telephone line.

33. The apparatus of claim 31, wherein the communication device comprises a cable modem, and the communication line comprises a cable line.

34. The apparatus of claim 31, wherein the communication device comprises a digital subscriber line modem, and the communication line comprises a digital subscriber line.

35. The apparatus of claim 31, wherein the communication device comprises an ISDN modem, and the communication line comprises an ISDN line.

36. The apparatus of claim 29, wherein the server executes gateway software to route network traffic to the Internet.

37. The apparatus of claim 36, wherein the gateway software comprises network address translation software.

38. The apparatus of claim 28, wherein the gateway software comprises proxy server software.

39. The apparatus of claim 29, where the local area network comprises network connections chosen from a set consisting of 10Base-T, 100Base-T, telephone lines, power lines, wireless infrared communications, and wireless RF communications.

* * * * *

US005696701A

United States Patent [19]

Burgess et al.

[11] Patent Number: 5,696,701

[45] Date of Patent: Dec. 9, 1997

[54] **METHOD AND SYSTEM FOR MONITORING THE PERFORMANCE OF COMPUTERS IN COMPUTER NETWORKS USING MODULAR EXTENSIONS**

[75] Inventors: **Gregory M. Burgess**, Kirkland, Wash.; **David B. Endicott**, Plano, Tex.; **Thomas Camarro**, Troy; **Richard C. Jagers**, Novi, both of Mich.

[73] Assignee: **Electronic Data Systems Corporation**, Plano, Tex.

[21] Appl. No.: **678,800**

[22] Filed: **Jul. 12, 1996**

[51] Int. Cl.[6] **G06F 11/00**

[52] U.S. Cl. **364/551.01**; 364/570; 364/550; 395/183.01; 395/183.02; 395/650; 395/200.01

[58] Field of Search ..... 364/551.01, 284.4, 364/264.5, 280, 265, 242.95, 264, 550, 514 B, 570; 395/650, 184.01, 500, 200.09, 200.01, 200.12, 200.03, 183.01, 183.02

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,780,821	10/1988	Crossley	364/200
4,949,248	8/1990	Caro	364/200
5,062,104	10/1991	Lubarsky et al.	370/60
5,202,989	4/1993	Hirosawa et al.	395/650
5,303,166	4/1994	Amalfitano et al.	364/551.01
5,367,670	11/1994	Ward et al.	395/575
5,402,431	3/1995	Saadeh et al.	371/67.1
5,459,837	10/1995	Caccavale	395/184.01
5,506,955	4/1996	Chen et al.	395/183.02
5,548,724	8/1996	Akizawa et al.	395/650
5,557,749	9/1996	Norris	395/200.18
5,581,482	12/1996	Wiedenman et al.	364/551.01
5,630,049	5/1997	Cardoza et al.	395/183.01

OTHER PUBLICATIONS

"Chapter 2 Zen and the Art of Performance Monitoring" from Optimizing Windows NT by Russ Blake, Copyright 1995, pp. 1 through 47 plus cover pages.

"Chapter 12 Writing a CustomWindows NT Performance Monitor" from Optimizing Windows NT by Russ Blake, Copyright 1995, pp. 1 through 14 plus cover pages.

"2.6.8 Performance Monitoring Service" from Optimizing Windows NT by Russ Blake, Copyright 1995, p. 1 plus cover pages.

"8.1 Performance Monitoring Service" from Optimizing Windows NT by Russ Blake, Copyright 1995, pp. 1 through 2 plus cover pages.

Primary Examiner—Emanuel T. Voeltz
Assistant Examiner—Bryan Bui
Attorney, Agent, or Firm—L. Joy Griebenow

[57] **ABSTRACT**

The invention comprises a method and system for monitoring the performance of a computer in a computer network using modular extension agents. In accordance with the method of the invention the first computer repeatedly obtains performance data including at least one performance value comprising a measure of the performance of the computer. The performance data is automatically sent over the computer network through a second computer coupled to the computer network. The second computer receives the performance data and passes the performance data to a first extension agent, wherein the first extension agent is operable to process the performance data.

20 Claims, 5 Drawing Sheets

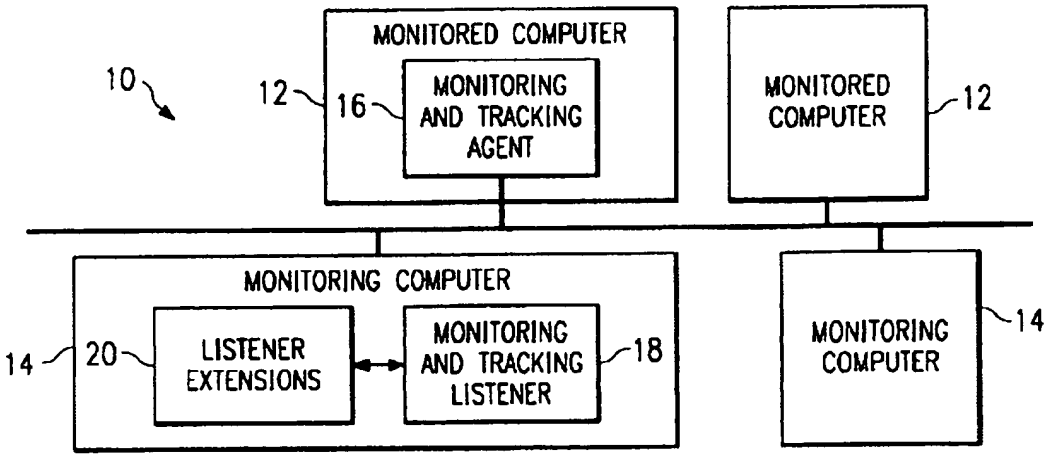


FIG. 1

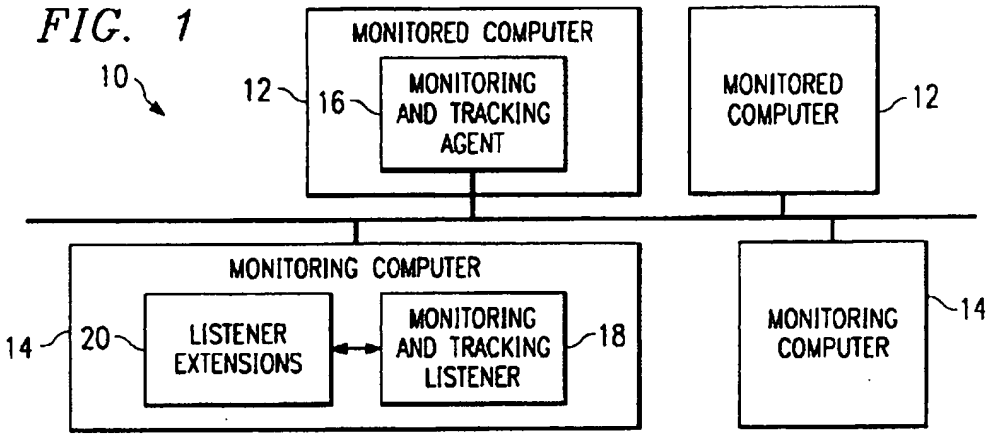

10
MONITORED COMPUTER
12
16
MONITORING AND TRACKING AGENT
MONITORED COMPUTER
12
MONITORING COMPUTER
14
20
LISTENER EXTENSIONS
MONITORING AND TRACKING LISTENER
18
MONITORING COMPUTER
14

FIG. 3

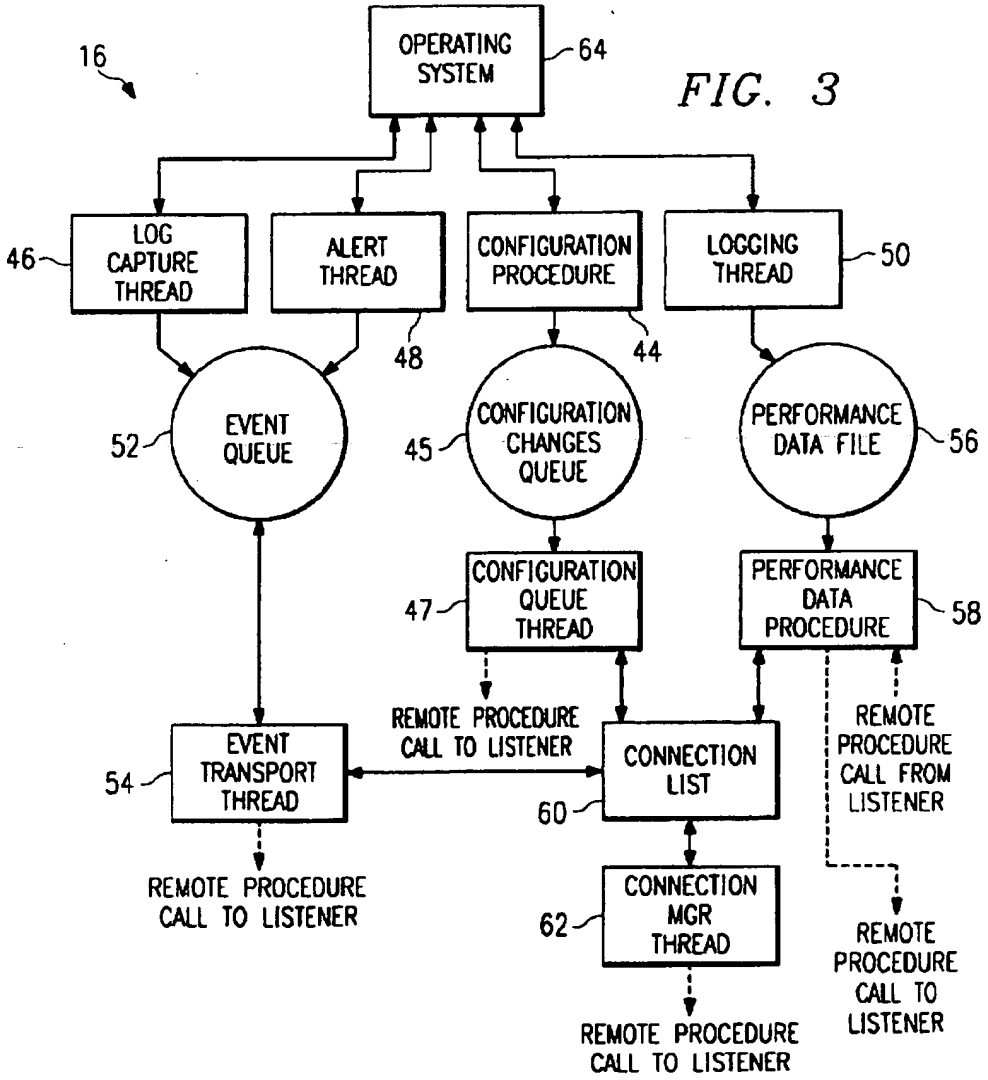

16
OPERATING SYSTEM
64
LOG CAPTURE THREAD
46
ALERT THREAD
CONFIGURATION PROCEDURE
LOGGING THREAD
50
48
44
EVENT QUEUE
52
CONFIGURATION CHANGES QUEUE
45
PERFORMANCE DATA FILE
56
CONFIGURATION QUEUE THREAD
47
PERFORMANCE DATA PROCEDURE
58
REMOTE PROCEDURE CALL TO LISTENER
REMOTE PROCEDURE CALL FROM LISTENER
EVENT TRANSPORT THREAD
54
CONNECTION LIST
60
CONNECTION MGR THREAD
62
REMOTE PROCEDURE CALL TO LISTENER
REMOTE PROCEDURE CALL TO LISTENER
REMOTE PROCEDURE CALL TO LISTENER

FIG. 2

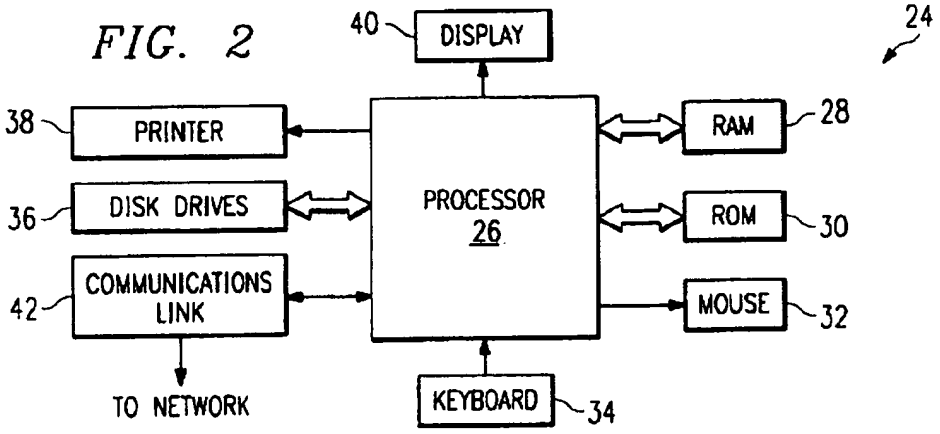
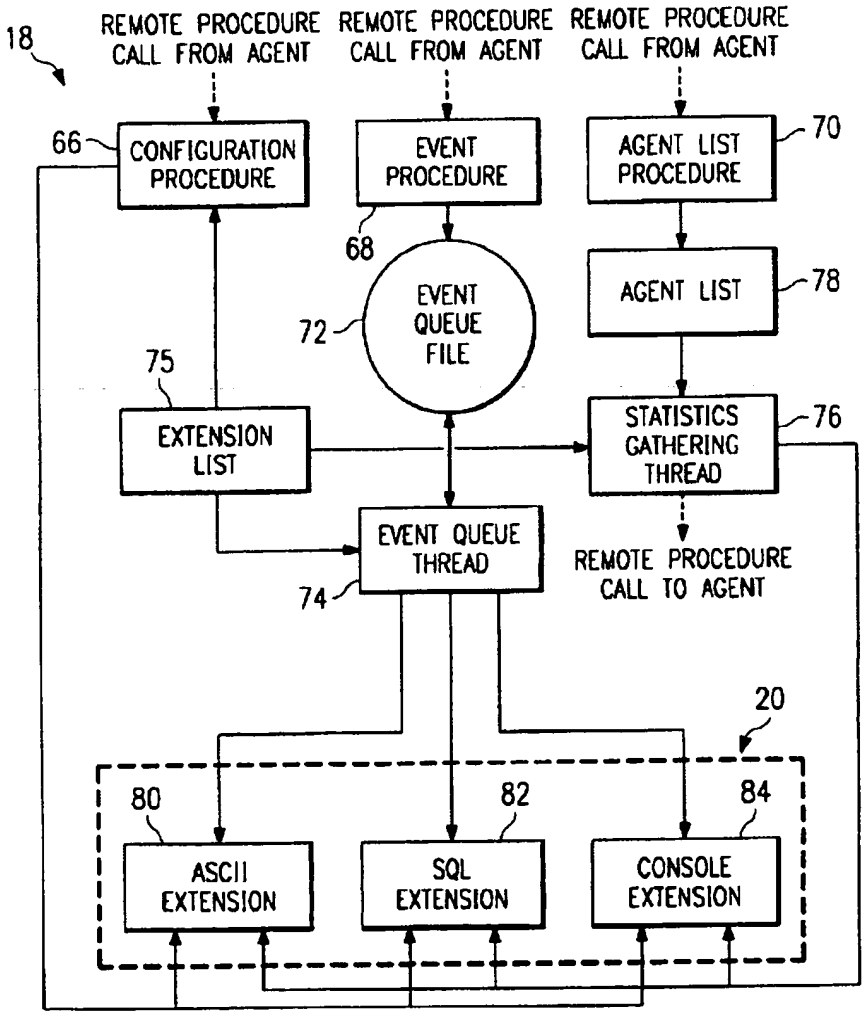

24
40
DISPLAY
38
PRINTER
36
DISK DRIVES
42
COMMUNICATIONS LINK
TO NETWORK
PROCESSOR 26
RAM
28
ROM
30
MOUSE
32
KEYBOARD
34

FIG. 4


18
REMOTE PROCEDURE CALL FROM AGENT
REMOTE PROCEDURE CALL FROM AGENT
REMOTE PROCEDURE CALL FROM AGENT
66
CONFIGURATION PROCEDURE
EVENT PROCEDURE
68
AGENT LIST PROCEDURE
70
EVENT QUEUE FILE
72
AGENT LIST
78
75
EXTENSION LIST
STATISTICS GATHERING THREAD
76
EVENT QUEUE THREAD
74
REMOTE PROCEDURE CALL TO AGENT
20
80
ASCII EXTENSION
82
SQL EXTENSION
84
CONSOLE EXTENSION

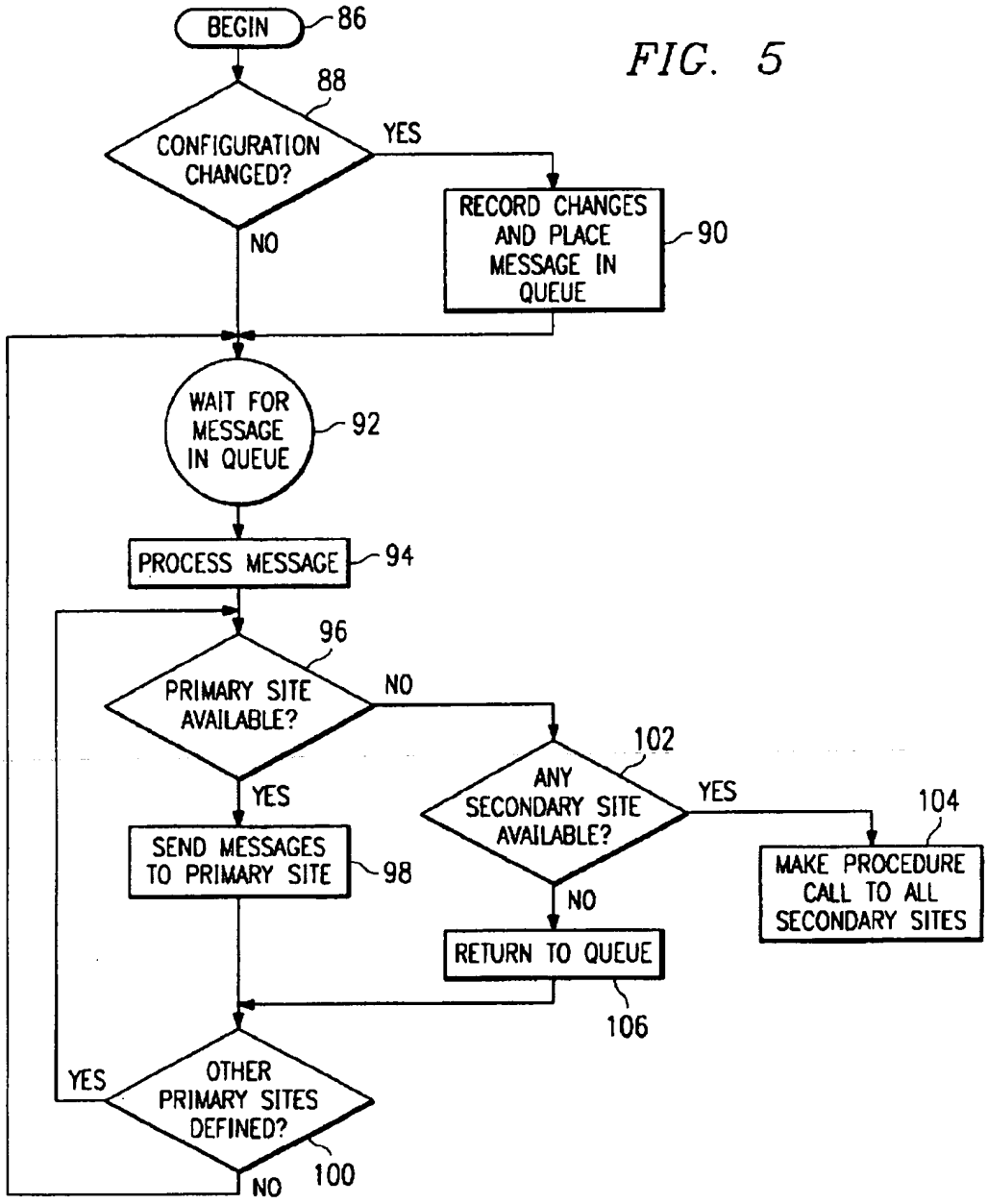

BEGIN
86
88
CONFIGURATION CHANGED?
YES
NO
RECORD CHANGES AND PLACE MESSAGE IN QUEUE
90
WAIT FOR MESSAGE IN QUEUE
92
PROCESS MESSAGE
94
96
PRIMARY SITE AVAILABLE?
NO
YES
SEND MESSAGES TO PRIMARY SITE
98
102
ANY SECONDARY SITE AVAILABLE?
YES
NO
104
MAKE PROCEDURE CALL TO ALL SECONDARY SITES
RETURN TO QUEUE
106
YES
OTHER PRIMARY SITES DEFINED?
NO
100

FIG. 5

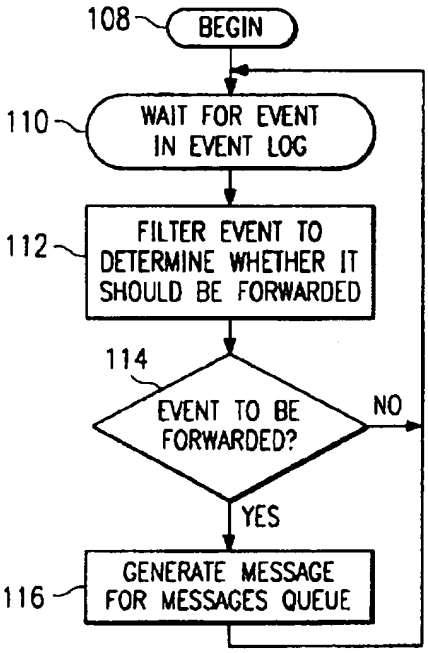


FIG. 6

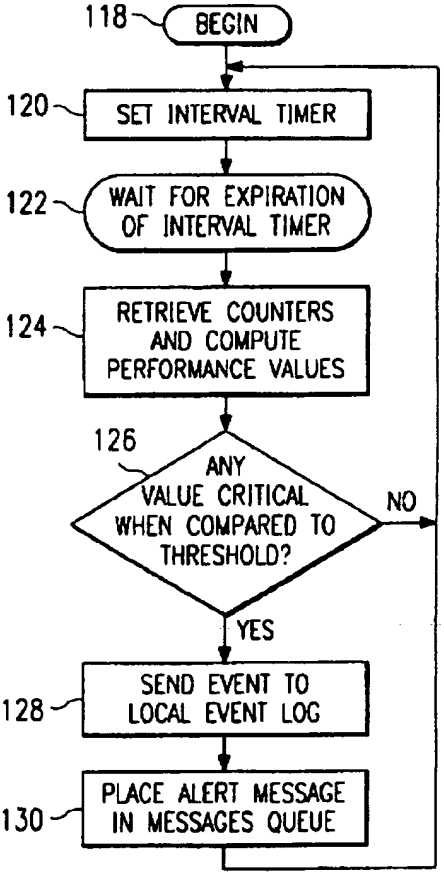


FIG. 7

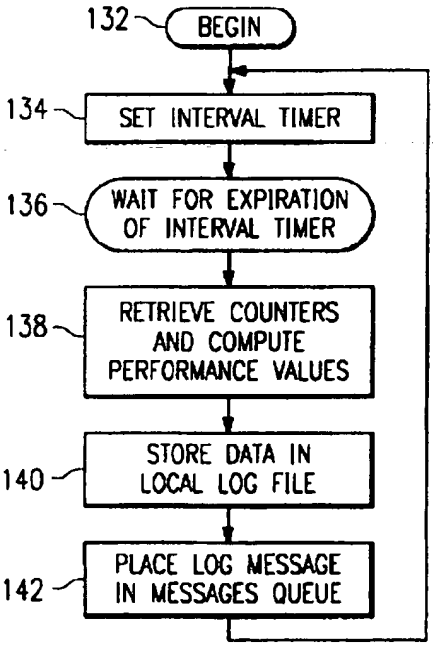


FIG. 8

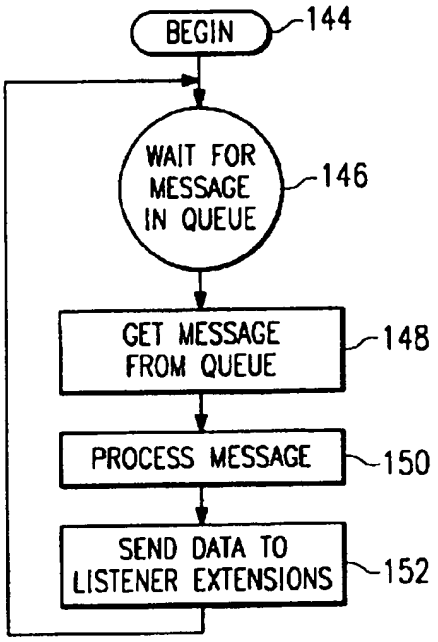

BEGIN
144
WAIT FOR MESSAGE IN QUEUE
146
GET MESSAGE FROM QUEUE
148
PROCESS MESSAGE
150
SEND DATA TO LISTENER EXTENSIONS
152

FIG. 9

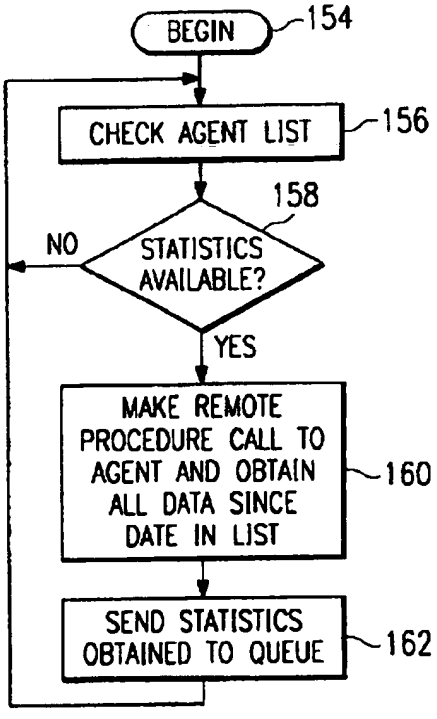

BEGIN
154
CHECK AGENT LIST
156
158
NO
STATISTICS AVAILABLE?
YES
MAKE REMOTE PROCEDURE CALL TO AGENT AND OBTAIN ALL DATA SINCE DATE IN LIST
160
SEND STATISTICS OBTAINED TO QUEUE
162

FIG. 10

METHOD AND SYSTEM FOR MONITORING THE PERFORMANCE OF COMPUTERS IN COMPUTER NETWORKS USING MODULAR EXTENSIONS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to U.S. application Ser. No. 08/678,934, filed on Jul. 12, 1996 by Gregory M. Burgess, et al. and entitled "Method and System for Performance Monitoring in Computer Networks," pending.

This application is related to U.S. application Ser. No. 08/679,293, filed on Jul. 12, 1996 by Gregory M. Burgess, et al. and entitled "Method and System for Tracking the Configuration of a Computer Coupled to a Computer Network," pending.

These applications have been commonly assigned to Electronic Data Systems Corporation.

TECHNICAL FIELD OF THE INVENTION

This invention relates generally to performance monitoring in computer networks and more particularly to a method and system for monitoring the performance of a computer in a computer network using modular extensions.

BACKGROUND OF THE INVENTION

The use of computer networks connecting a number of computers has been increasing. Often, computer networks are connected in a client server relationship. One or more server computers normally contain resources that are shared among a number of client computers. Shared resources can also be stored on client computers.

Because a server is a shared resource, the capabilities that the server needs to possess to adequately serve the client computers vary depending upon the number of users of the shared resources. As the number of client computers grows, demands on a server will often increase. The capabilities of the server may affect the amount of time it takes for a client to access a shared resource on a server. If the server does not have sufficiently high performance characteristics, clients may incur greater delays in waiting to access shared resources. Network planners thus take server performance into account when designing a network and when upgrading a network.

Depending upon the configuration of a network, the types of tasks ordinarily performed by users of the network, usage patterns that may develop over the course of a workday, and other such variables, the performance level desired for a particular server in a particular computer network may vary considerably. One way to determine the performance desired for a particular server in a particular network is to study the performance of existing servers on that network over time. By gathering performance statistics, network planners can better determine the performance level desirable for upgrades either to specific servers or to the network as a whole such as by adding additional servers.

Network planners will normally use some type of performance monitoring software to aid in the task of monitoring computer performance. Unfortunately, existing performance monitoring software normally requires human intervention. A network planner often must be either physically present at the computer whose performance he desires to monitor or logged onto that computer from a remote location. Thus, performance monitoring may be time consuming because a network operator will normally need to be physically present when using performance monitoring software. In a large network, a network operator might spend a day or several days simply obtaining a small amount of performance data from each server in the network.

Existing performance monitoring software normally has a number of built-in functions. Adding additional functions desired by a particular network operator may be difficult and expensive. Existing programs that require network operator intervention are also not useful in detecting server performance that has become so critical that the server may fail or that the entire network may fail. This performance data might only be obtained if the network operator happens to be monitoring the performance of the particular server at the time when its performance becomes critical. Existing performance monitoring software, therefore, is mostly unhelpful in predicting either server or network failure.

Finally, existing performance monitoring programs do not allow easy tracking of the configuration of computers in the network.

SUMMARY OF THE INVENTION

The invention comprises a method and system for monitoring the performance of a computer coupled to a computer network. The invention employs a novel architecture including modular program extensions to allow flexible upgrades to the invention's performance monitoring capability. One aspect of the invention is a method for monitoring the performance of a first computer coupled to a computer network. Performance data is repeatedly obtained using the first computer and comprises at least one performance value which is a measure of the performance of the first computer. The performance data is automatically sent over the computer network to a second computer coupled to the computer network. The performance data is received at the second computer and then passed to a first extension agent wherein the first extension agent is operable to process the performance data.

The invention has several important technical advantages. The invention allows simultaneous performance monitoring of a number of computers, typically server computers, connected to a computer network. The performance data is sent to a central listener application which allows the performance data to be easily logged for future reference by a network administrator. The modular extension agent architecture of the invention allows the performance monitoring capabilities of the central listener application to be easily upgraded to include additional capabilities. Also, because the central listener application is modular, different embodiments of the invention with different capabilities may be sold as different products to different customers. In addition, in certain networks, the network administrator may only desire to provide some of the possible performance monitoring capabilities associated with the invention. The modularity of the central listener application allows a network administrator to choose which performance monitoring capabilities to provide at particular locations.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention and the advantages thereof, reference is now made to the following descriptions taken in conjunction with the accompanying drawings in which:

FIG. 1 illustrates a computer network using the method and system of the present invention;

FIG. 2 illustrates a computer that may be used in the computer network of FIG. 1;

FIG. 3 illustrates a block diagram of a monitoring and tracking agent constructed in accordance with the teachings of the invention;

FIG. 4 illustrates a block diagram of a monitoring and tracking listener with listener extensions constructed in accordance with the teachings of the invention;

FIG. 5 illustrates a flow chart of the operation of the monitoring and tracking agent of FIG. 3;

FIG. 6 illustrates a flow chart of the operation of the event capture thread of the monitoring and tracking agent of FIG. 3;

FIG. 7 illustrates a flow chart of the operation of the alert thread of the monitoring and tracking agent of FIG. 3;

FIG. 8 illustrates a flow chart of the operation of the logging thread of the monitoring and tracking agent of FIG. 3;

FIG. 9 illustrates a flow chart of the operation of the monitoring and tracking listener application of FIG. 4; and

FIG. 10 illustrates a flow chart of the operation of the statistics gathering thread of the monitoring and tracking listener application of FIG. 4.

DETAILED DESCRIPTION OF THE INVENTION

The preferred embodiment of the present invention and its advantages are best understood by referring to FIGS. 1–10 of the drawings, like numerals being used for like and corresponding parts of the various drawings.

FIG. 1 illustrates a computer network **10** that employs the performance monitoring method and system of the present invention. Computer network **10** comprises a plurality of monitored computers **12** networked to a number of monitoring computers **14**. Although this embodiment has two monitored computers **12** and two monitoring computers **14**, other embodiments of the invention may have either a different number of monitored computers **12** or monitoring computers **14**. Ordinarily, the monitored computers **12** will be server computers but could be any type of computer. Computer network **10** may comprise any type of computer network such as a local area network or wide area network.

Monitoring and tracking agent **16**, which comprises a program that is run by monitored computer **12**, monitors the performance of monitored computer **12**. Monitoring and tracking agent **16** sends performance data to monitoring and tracking listener **18**, which is a program running on monitoring computer **14**. Monitoring and tracking listener **18** may, in turn, forward the data obtained from monitoring and tracking agent **16** to one or more listener extensions **20**, which comprise programs running on monitoring computer **14**.

Although the structure and operation of monitoring and tracking agent **16** and monitoring and tracking listener **18** will be described in more detail below, a brief overview of the operation of each will now be given. Monitoring and tracking agent **16** will normally run as an operating system service on monitored computer **12**. In this embodiment, monitoring and tracking agent **16** is a WINDOWS NT service. Monitoring and tracking agent **16** may perform several functions related to performance monitoring of monitored computer **12**. Monitoring and tracking agent **16** monitors .the performance of monitored computer **12** over time and captures performance data reflecting one or more aspects of the performance of monitored computer **12**. At preset intervals, it stores the data in a log file and notifies monitoring and tracking listener **18** that it has new data. When monitoring and tracking listener **18** is ready to gather the performance data, it retrieves previously ungathered performance data from monitoring and tracking agent **16**.

Monitoring and tracking agent **16** also generates alerts if one or more aspects of the performance of monitored computer **12** has reached a critical level. Monitoring and tracking agent **16** monitors the performance of monitored computer **12** at preset intervals and compares one or more performance parameters to preset thresholds. If the comparison to a threshold indicates that performance has become alertable, monitoring and tracking agent **16** generates an event which is sent to the local operating system event log of monitored computer **12** and sends a message to monitoring and tracking listener **18** with the information regarding the alert.

Monitoring and tracking agent **16** also monitors operating system events on monitored computer **12**. Each time a new event is placed in the operating system event log (the WINDOWS NT event log in this embodiment), monitoring and tracking agent **16** processes the event. If the user of monitoring and tracking agent **16** has indicated that this type of event has a high enough priority to monitor directly, then monitoring and tracking agent **16** sends a message with the event information to monitoring and tracking listener **18**. Low priority events are filtered out and ignored.

Monitoring and tracking agent **16** may also be used to monitor configuration changes for monitored computer **12**. Each time that monitoring and tracking agent **16** is started, it analyzes the current system configuration of monitored computer **12** to determine whether any changes have been made since the prior time that monitoring and tracking agent **16** was started. If changes have been made, these changes are sent to monitoring and tracking listener **18** so that monitoring computer **14** may keep track of the most current configuration of monitored computer **12**.

Monitoring and tracking listener **18** will normally be run as an operating system service on monitoring computer **14**. In this embodiment, monitoring and tracking listener **18** comprises a WINDOWS NT service running on monitoring computer **14**. Monitoring and tracking listener **18** receives messages and data from monitoring and tracking agent **16**. Monitoring and tracking listener **18** also dispatches data and messages to one or more plug-in listener extensions **20**. Listener extensions **20** may then direct some or all of the data to databases, a central console, or simple text files. Monitoring and tracking listener **18** could also perform some of these functions itself rather than handling these functions using listener extensions **20**.

FIG. 2 illustrates a general purpose computer **24** that may be used for either monitored computer **12**, monitoring computer **14**, or both. Computer **24** may be adapted to execute any of the well known MSDOS, PCDOS, OS2, UNIX, MOTIF, MAC-OS, X-WINDOWS, and WINDOWS operating systems, or other operating systems. Computer **24** comprises processor **26**, random access memory (RAM) **28**, read only memory (ROM), **30**, mouse, **32**, keyboard **34** and input/output devices such as printer **38**, disk drives **36**, display **40** and communications link **42**. The present invention includes computer software that may be stored in RAM **28**, ROM **30**, or disk drives **36** and is executed by processor **26**. Communications link **42** is connected to computer network **10**, but could also be connected to a telephone line, an antenna, a gateway, or any other type of communications link. Disk drives **36** may include a variety of types of storage media such as, for example, floppy disk drives, hard disk drives, CD-ROM drives, or magnetic tape drives.

FIG. 3 illustrates a block diagram of an embodiment of monitoring and tracking agent 16 constructed in accordance with the invention. Monitoring and tracking agent 16 is preferably a multi-threaded application that can efficiently process performance data. The structure of monitoring and tracking agent 16 illustrated in FIG. 3 is only one of many possible structures and other structures could be used to perform the functions of the invention. In addition, monitoring and tracking agent 16 need not be a multi-threaded application.

Monitoring and tracking agent 16 comprises configuration procedure 44, log capture thread 46, alert thread 48, logging thread 50, event transport thread 54, configuration queue thread 47, performance data procedure 58 and connection manager thread 62. Monitoring and tracking agent 16 communicates with operating system 64. Operating system 64 comprises the operating system for monitored computer 12. In addition, monitoring and tracking agent 16 maintains event queue 52, performance data file 56, and connection list 60. Although the operation of monitoring and tracking agent 16 will be more fully described in connection with FIGS. 5–8 below, a description of the operation of each component is now provided.

Configuration procedure 44 is invoked by a main thread (not explicitly shown), which handles initialization as well as configuration change reporting. When monitoring and tracking agent 16 is started, configuration procedure 44 gathers system configuration information such as information about the operating system, hardware, software, memory, logical disks, system environment, services and drivers, and install devices of monitored computer 12. This information can be used to track configuration changes such as software updates and hardware upgrades for monitored computer 12. Configuration procedure 44 also starts configuration queue thread 47. Configuration procedure 44 obtains the configuration information from operating system 64. In this embodiment, configuration procedure 44 obtains the configuration information from the WINDOWS NT registry file maintained by operating system 64. Configuration procedure 44 compares the obtained configuration information with prior configuration information that configuration procedure 44 maintains in a file (not explicitly shown). If the configuration information is unchanged, configuration procedure 44 simply terminates. If the configuration information has changed, configuration procedure 44 generates a message for monitoring and tracking listener 18 containing the configuration information. Configuration procedure 44 places this message in configuration changes queue 45. Configuration queue thread 47 reads configuration changes out of configuration changes queue 45 and sends to listeners appearing on connection list 60 as described below.

Examples of configuration information that may be obtained by configuration procedure 44 regarding the operating system include the date of installation, the registered owner, the registered organization, the current version, the system root directory, the current version of operating system, the product type, the build number of the operating system, and various start-up options of the operating system. Examples of configuration information that configuration procedure 44 may obtain regarding the hardware include the OEM ID, the system BIOS date, the page file size for virtual memory, the minimum application address, the maximum application address, the processor type, and number of processors. Examples of configuration information that configuration procedure 44 may obtain regarding the memory include the current load, the total physical memory, the available physical memory, the total page file, the available page file, the total virtual memory size, and the available virtual memory size. Configuration procedure 44 may also obtain configuration information regarding the services and drivers available on the system. Examples of the configuration information about each service or driver include the name of the service or driver, its status, the service type, the start-up configuration, the full path name of the executable image, the file size, the file date, the version number, the location of the raw version, and the location of the ASCII version.

Log capture thread 46 is used to monitor events occurring on monitored computer 12. Log capture thread 46 processes each event received by an event log associated with operating system 64. In this embodiment, the event log is the WINDOWS NT event log. Log capture thread 46 monitors the event log for new events. If a new event is posted to the event log, log capture thread 46 will process that event.

Log capture thread 46 provides a filtering feature when processing events. If an event is filtered out, log capture thread 46 ignores that event. If an event was not filtered out, then log capture thread 46 generates a message indicating that an event has occurred, wherein the message includes data about the event. Log capture thread 46 places the message in event queue 52. The message will eventually be forwarded to monitoring and tracking listener 18.

In this embodiment, operating system 64 has multiple event logs that are monitored by log capture thread 46. Each event log obtains events from specific sources in monitored computer 12. Each event also is associated with an event identification number identifying particular events. Log capture thread 46 may filter events based upon the event log in which they appear, the source of the event, and/or the event identification number. Filters can include or exclude specific event identification numbers, event sources, or event logs. Filtering is hierarchical in this embodiment: event identification first, event source second, and event log last. A flag may be used to indicate whether to include or exclude events with a specific identification number, source, or event log.

Alert thread 48 monitors various performance counters reflecting the performance of monitored computer 12, calculates performance values for those counters and compares those values to predetermined thresholds. If the comparison indicates that the performance of monitored computer 12 has reached an alertable level, then alert thread 48 generates an alert. Alerts generated by alert thread 48 comprise a message identifying the message as an alert and including data about the alert. Alert thread 48 places such alert messages in event queue 52.

Alert thread 48 obtains performance information about monitored computer 12 using counters maintained by operating system 64. Each counter reflects one aspect of the performance of monitored computer 12. In this embodiment, the counters comprise WINDOWS NT performance counters maintained by the WINDOWS NT operating system. Alert thread 48 monitors these counters at an adjustable time interval which is determined by the user of monitoring and tracking agent 16. After the passage of each interval of time, alert thread 48 captures the values of the performance counters, calculates performance values based upon the counters, and compares the performance values to predetermined threshold values determined by the user of monitoring and tracking agent 16. A performance value might be the value of a performance counter itself. The user of monitoring and tracking agent 16 may associate various levels of performance of a particular performance value with different levels of alerts such as harmless, medium severity, or

critical. The transmission of an alert indicates that the performance of monitored computer **12** has reached an alertable level.

Alert thread **48** may monitor many different performance counters. For example, alert thread **48** may monitor the percentage of free space remaining on each logical disk associated with monitored computer **12**. In this embodiment, an alert is generated when the percentage of free space is less than five percent. This alert provides a warning that a work station or server logical drive is becoming full and could prevent users from saving data and/or cause performance degradation.

In this embodiment, alert thread **48** also monitors the percentage of time that a processor is busy executing a non-idle thread. This performance data could be viewed as a measure of the percentage of time that the processor of monitored computer **12** spends doing useful work. Each processor of monitored computer **12** is assigned an idle thread in the idle process which consumes unused processor cycles. This alert may identify sustained high processor utilization which may indicate that a process is running improperly or that the processor is approaching capacity. In this embodiment, an alert is generated if the percentage of time that a processor is busy executing a non-idle thread exceeds 85 percent.

Alert thread **48**, in this embodiment, also monitors the rate at which the operating system switches between threads. Thread switches can occur either inside a single process or across multiple processes. A thread switch is caused either by another thread requesting processor time or by a higher priority thread preempting the current thread. Excessive context switches between threads may indicate that the operating system of monitored computer **12** is overburdened or causing performance degradation. In this embodiment, an alert is generated if the number of context switches per second exceeds 2,000.

Alert thread **48**, in this embodiment, also monitors the percentage of free space available in each paging file. As the percentage of free space decreases, this performance value may indicate that the operating system is approaching the limits of its virtual memory. If the paging files are full, the operating system of monitored computer **12** may halt. Monitoring of this performance value may allow a command to monitored computer **12** to increase the size of its paging file to accommodate the operating system demands. In this embodiment, if the paging files are over 80 percent full, an alert is generated.

Alert thread **48** also monitors the amount of physical memory available to the operating system. If a process running on monitored computer **12** requires more RAM than is available, the operating system must swap another process out to the paging files to free memory, which may slow system performance. As the availability of physical memory shrinks, the operating system must swap more processes in and out of memory and performance may degrade. Monitoring of this performance data may allow a message to be sent to monitored computer **12** to cease executing extraneous processes. In this embodiment, an alert is generated if the available physical memory falls below one megabyte.

Alert thread **48** also monitors the number of times that the operating system is not able to assign a work item to service a request. If a process is not assigned a work item, it will not be serviced by the operating system and must wait before proceeding. Monitoring of this performance data may allow a message to be sent to monitored computer **12** to tune the operating system to provide more work items to service requests. In this embodiment, if the number of work item shortages is greater than 30, an alert is generated.

Alert thread **48** also monitors the number of times an internal server error was detected. Under normal circumstances server errors should not occur. Unexpected errors usually indicate a problem with the operating system and can result in an operating system halt. Monitoring of this performance information may allow a network administrator to take action before the halt of the operating system of monitored computer **12**. In this embodiment if the number of system errors exceeds ten, then an alert is generated.

Logging thread **50** handles logging of performance data. Logging thread **50** logs performance data each predetermined time interval. The predetermined time interval may be set by the user of monitoring and tracking agent **16** and may or may not be the same as the interval over which alert thread **48** monitors performance for purposes of generating alerts. After the passage of the predetermined time interval, logging thread **50** obtains the values of performance counters from operating system **64**. From this data, logging thread **50** may calculate performance values based upon the counters. Some of the counters may also comprise performance values without further calculation. Logging thread **50** stores the relevant performance data with a time stamp identification in performance data file **56**. Performance data file **56** is a data file that may be stored on the disk drives **36** of monitored computer **12**. After obtaining the performance data and storing it in performance data file **56**, logging thread **50** causes data to be placed in connection list **60** indicating the date and time at which the performance data was captured. If this time and date is later than an existing entry, than the existing entry is maintained.

In this embodiment, logging thread **50** monitors a number of performance values. The user of monitoring and tracking agent **16** may choose which performance values to monitor and may disable the monitoring of others. In this example, performance values monitored by logging thread **50** include the available bytes of memory, the cache bytes available, the committed bytes, the number of page faults per second, the number of cache faults per second, the peak number of cache bytes, the number of pages per second, the size of the paged bytes pool, and the size of the non-paged bytes pool. Logging thread **50** also monitors performance data relating to the operating system of monitored computer **12**. Examples of such performance values include the percentage of total operating system privilege time, the percentage of total processing time, the percentage of total user time, the number of context switches per second, the number of file control bytes per second, the number of file control operations per second, the number of file data operations per second, the number of file read bytes per second, the number of file read operations per second, the number of file write bytes per second, the number of file write operations per second, the number of system calls per second, the total number of interrupts per second, and the system up time.

Logging thread **50** also monitors performance data relating to the physical disks of monitored computer **12**. Examples of performance values monitored by logging agent **50** include the percentage of time devoted to disk access, the percentage of time devoted to disk reads, and the percentage of time devoted to disk writes. Similar counters may be monitored for each logical disk and the percentage of free space on each logical disk may also be monitored. The percentage usage of the paging file may also be monitored by logging thread **50**.

Several performance values may also be monitored which are known as server parameters. Logging thread **50** monitors

the number of work item shortages, the number of server sessions, the number of pool non-paged failures, the number of pool paged failures, and the number of system errors. Other performance values may be monitored without departing from the scope of the invention.

Because performance data is considered lower priority than event, alert, and configuration data, the invention employs a mechanism such that monitoring and tracking listener **18** retrieves the performance data from performance data file **56** when it is not performing higher priority functions. In this embodiment, monitoring and tracking agent **16** and monitoring and tracking listener **18** work in a push-pull configuration wherein monitoring and tracking agent **16** notifies listener **18** that new performance data is available (push), and monitoring and tracking listener **18** retrieves the performance data when it has completed higher priority tasks (pull). If the agent has new performance data and the listener has not yet responded to a previous notification, the agent in this embodiment will not send another notification. In this embodiment, a date and time stamp is sent with each notification so subsequent notifications are not sent. Monitoring and tracking listener **18** collects all outstanding data when it responds to a request by collecting all data placed in performance data file **56** having a date and time stamp equal to or after the date and time stamp in the notification received from monitoring and tracking agent **16**.

The notification by monitoring and tracking agent **16** that performance data is available is sent to monitoring and tracking listener **18** by making a remote procedure call to agent list procedure **70**. Dispatcher thread **54** retrieves the message from event queue **52** and makes a remote procedure call over the network using a TCP/IP protocol to the agent list procedure of monitoring and tracking listener **18**. When monitoring and tracking listener **18** desires to obtain the performance data from monitoring and tracking agent **16**, it generates a remote procedure call of its own to performance data procedure **58** of monitoring and tracking agent **16**. Performance data procedure **58** obtains the performance data from performance data file **56** and returns the data to monitoring and tracking listener **18**. Performance data file **56** may be stored and/or sent over the network in compressed form.

Event queue **52** is a file that may be stored on a disk drive **36** of monitored computer **12**. Event queue **52** obtains messages intended to be sent to monitoring and tracking listener **18**. Event transport thread **54** handles the sending of messages in event queue **52** to the appropriate monitoring and tracking listener **18**. Event queue **52** may comprise a single queue or a plurality of queues associated with the various threads of monitoring and tracking agent **16**. Event transport thread **54** passes the messages to monitoring and tracking listener **18** by generating appropriate remote procedure calls over computer network **10** using a TCP/IP protocol.

The invention allows messages and data to be sent to one or more monitoring computers **14**. A monitoring computer **14** may be classified as a primary or secondary site. In this embodiment, monitoring and tracking agent **16** sends all events, log data, and configuration data to a primary monitoring computer **14**. Monitoring and tracking agent **16** will send events and configuration data to secondary monitoring computers **14** if a primary monitoring computer **14** is not available. In this embodiment, however, performance data that was logged using logging thread **50** is queued until the primary monitoring computer **14** is available.

The use of multiple monitoring computers **14** is accomplished using connection manager thread **62**. Connection manager thread **62** maintains a list of all monitoring computers **14** to which monitoring and tracking agent **16** will report data. The list includes both primary and secondary monitoring computers **14**. Connection manager thread **62** maintains the status of each monitoring computer **14** as being either on-line or off-line. The status of each monitoring computer **14** is stored in connection list **60**. Connection list **60** comprises a file that may be stored on a disk drive **36** of monitored computer **12**. Connection list **60** may also be stored in the RAM **28** of monitored computer **12**.

Connection manager thread **62** determines whether a monitoring computer **14** is on-line or off-line by generating remote procedure calls to the monitoring and tracking listener **18** of that monitoring computer **14**. If the appropriate response is received, then connection manager thread **62** records the status of that monitoring computer as being on-line. If no response is received, then connection manager thread **62** records the status of that monitoring computer **14** as being off-line.

When an event, alert, or configuration data is to be sent, event transport thread **54** consults connection list **60**. If a primary site cannot be reached, event transport thread **54** will attempt to send the message and/or data to one or more secondary sites associated with that primary site. If at least one secondary site can be reached, then the message is sent to that secondary site. If a secondary site cannot be reached, then the message will remain in event queue **52** until a connection to either the primary or secondary monitoring computer **14** can be established. Note that multiple primary monitoring computers **14** may be designated for a particular monitoring and tracking agent **16**. Thus, monitored computer **12** may report alerts to multiple locations. This feature of the invention is particularly useful in large computer networks where network administrators may be monitoring network performance at various locations.

When monitoring and tracking agent **16** has new performance data to send, it will consult connection list **60** only for the list of primary monitoring computers **14**. If a particular primary monitoring computer **14** cannot be contacted, the notification will remain in connection list **60** until that particular primary monitoring computer **14** is available.

FIG. **4** illustrates an embodiment of monitoring and tracking listener **18** constructed in accordance with the invention. The structure of monitoring and tracking listener **18** is only one example of how monitoring and tracking listener **18** could be structured to perform the methods of the invention. In addition, monitoring and tracking listener **18** is a multi-threaded application but need not be a multi-threaded application.

Monitoring and tracking listener **18** comprises configuration procedure **66**, event procedure **68**, agent list procedure **70**, statistics gathering thread **76**, and event queue thread **74**. In addition, monitoring and tracking listener **18** maintains extension agent event queue file **72**, extension list **75**, and agent list **78**. Monitoring and tracking listener **18** also is coupled to one or more listener extensions **20**. In this embodiment, monitoring and tracking listener **18** is coupled to ASCII extension agent **80**, SQL database extension agent **82**, and console extension agent **84**. The operation of monitoring and tracking listener **18** is described in more detail in connection with FIGS. **9** and **10** below but the operation of components of monitoring and tracking listener **18** will now be partially described. Configuration procedure **66** receives remote procedure calls from one or more monitoring and tracking agents **16**. Configuration procedure **66** receives remote procedure calls which are messages containing con-

figuration data regarding the configuration of a monitored computer 12. Configuration procedure 66 processes the message and forwards the configuration data to each listener extension 20.

Event procedure 68 receives remote procedure calls from one or more monitoring and tracking agents 16 which are messages regarding events or alerts detected by monitoring and tracking agent 16. Event procedure 68 processes the events and alerts and forwards the data regarding each event or alert to event queue file 72. Agent list procedure 70 maintains a list of all monitored computers 12 for which the monitoring computer 14 running monitoring and tracking listener 18 is a primary monitoring computer 14. Agent list procedure 70 maintains the list as agent list 78 which may be stored in the RAM 28 of monitoring computer 14 or on a disk drive 36 of monitoring computer 14. Agent list 78 contains an indication as to whether each monitored computer 12 has performance data to be logged by monitoring and tracking listener 18 as well as a time stamp indicating the time and date of the oldest performance data obtained by that agent that has not been retrieved by monitoring and tracking listener 18.

Agent list procedure 70 receives notifications from monitoring and tracking agent 16 along with the time and date stamp associated with the notification indicating that a particular monitored computer 12 has performance data available to be logged. Agent list procedure 70 then updates agent list 78. When monitoring and tracking listener 18 is not busy (i.e. there are free processor cycles to service the statistics gathering thread), then statistics gathering thread 76 will obtain performance data for each agent in the agent list 78 that has a pending notification that performance data is available. Statistics gathering thread 76 will obtain the performance data by generating a remote procedure call to performance data procedure 58 of monitoring and tracking agent 16. Statistics gathering thread 76 will obtain all performance data that has been logged by the corresponding monitoring and tracking agent 16 since the time and date stamp associated with that agent in agent list 78. After statistics gathering thread 76 has obtained the performance data, it passes each instance of the performance data to each of the listener extensions 20.

Event queue thread 74 dispatches the messages and data received by monitoring and tracking listener 18 to one or more listener extensions 20 coupled to monitoring and tracking listener 18. Event queue thread obtains a list of extensions from extension list 75. Extension list 75 keeps a dynamic list of available listener extensions 20 and event queue thread 74, statistics gathering thread 76, and configuration procedure 66 all send data to the extensions on the list. In this embodiment, listener extensions 20 comprise dynamic link library procedures. This feature of the invention allows monitoring and tracking listener 18 to be modularized. When a network administrator desires additional functionality to process data received by monitoring and tracking listener 18, the network administrator can simply create another listener extension 20 and cause the data received by monitoring and tracking listener 18 to be sent to that listener extension. In this embodiment, event queue thread 74 dispatches data and messages in event queue file 72 to each of the listener extensions 20.

ASCII extension agent 80 comprises a procedure that takes the data obtained from event queue thread 74, statistics gathering thread 76, and configuration procedure 66 and writes the data to a text file. The text file may be stored on the disk drive 36 of monitoring computer 14 or may be stored at a remote location.

SQL database extension agent 82 receives the data from event queue thread 74, statistics gathering thread 76, and configuration procedure 66 and stores the data in an SQL database, which also could be stored on the disk drive 36 of monitoring computer 14 or at a remote location. Storing the data in an SQL database allows further processing by any open database connectivity (ODBC) compliant database software.

Console extension agent 84 processes the data obtained from event queue thread 74, statistics gathering thread 76, and configuration procedure 66 and forwards some or all of that data to a network monitoring console. In this embodiment, console extension 84 forwards events and alerts to an enterprise console such as that available from Tivoli. The enterprise console comprises an event correlator that takes events from network management systems and combines them in a common format. Console extension agent 84 queues events before sending them to the enterprise console. The events may be sent to the enterprise console over computer network 10 using communications link 42 of monitoring computer 14. Alternatively, communications link 42 may have a direct connection to the monitoring console. Console extension agent 84 may also send all events to multiple monitoring consoles. The events may be filtered by console extension agent 84 such that only the most important events are forwarded to the enterprise console. Any type of monitoring console could be used without departing from the scope of the invention.

In this embodiment, event queue file 72 comprises a single queue. Event queue thread 74, statistics gathering thread 76, and configuration procedure 66 seek to send data to the various listener extensions at a constant rate. Sometimes, one of these threads may have to wait for an extension as the various extensions may process data at different rates. Each listener extension 20 may maintain its own queue of data to compensate for a variance in processing speed.

FIG. 5 illustrates a flow chart of the general operation of monitoring and tracking agent 16. The process begins in step 86 with initialization. In step 88, configuration procedure 44 determines whether the configuration of the monitored computer 12 has changed. If so, then the changes are recorded in step 90 and a message is placed in configuration changes queue 45. Following step 88, if the configuration was not changed, or following step 90 if the configuration was changed, the procedure continues at step 92 where event transport thread 54 waits for messages to appear in event queue 52. When a message appears in event queue 52, event transport thread 54 processes the message in step 94.

Following the initial processing of the message from event queue 52, event transport thread 54 accesses connection list 60 in step 96 to determine whether a primary site is available or not. If so, then the proper message is sent to the monitoring and tracking listener 18 of that primary site in step 98. Event transport thread 54 generates the proper remote procedure call to the monitoring and tracking listener 18 and sends the data as described above. Then, in step 100, it is determined whether there are other primary sites defined. If so, then the process continues in step 96. If not, then the process continues in step 92, where event transport thread 54 waits for another message to appear in the queue.

If a primary site was not available in step 96, then in step 102 it is determined whether a secondary site is available. Event transport thread 54 determines whether a secondary site is available by accessing connection list 60. If no secondary site is available, then the message is returned to

the queue in step 106 with a notation that the message could not be sent to the particular primary and secondary sites. The process then continues in step 100.

If any defined secondary site was available in step 102, then the procedure continues in step 104. Event transport thread 54 makes the appropriate remote procedure calls to all available secondary sites for the primary site that was not available in step 96. In this embodiment, only one secondary site need be available to avoid returning a message to the queue. Thus, in step 104, event transport thread 54 makes the appropriate remote procedure calls to all available secondary sites for the unavailable primary site. Following step 104 the process continues in step 100 where it is determined whether any other primary sites are defined.

FIG. 6 illustrates a flow chart describing the operation of an embodiment of log capture thread 46. The process begins in step 108 with initialization of log capture thread 46. In step 110, log capture thread 46 waits for an event to appear in the event log of operating system 64. When an event is received, the process continues in step 112. Log capture thread 46 filters the event to determine whether it should be forwarded or not. In step 114, it is determined whether the event should be forwarded or not to monitoring and tracking listener 18. If not, then the process continues in step 110 with log capture thread 46 waiting for another event. If the event is to be forwarded, then in step 116 an appropriate message is generated and placed in event queue 52. Log capture thread 46 continues this process until monitoring and tracking agent 16's execution is terminated.

FIG. 7 illustrates a flow chart describing the operation of one embodiment of alert thread 48. The process begins at step 118 with initialization of alert thread 48. In step 120, an interval timer is set to a predefined interval defined by the user of monitoring and tracking agent 16. Then, in step 122, alert thread 48 waits for the expiration of the interval timer. When the interval timer expires, alert thread 48 retrieves performance counters maintained by operating system 64 and computes performance values based upon those performance counters. In step 126, alert thread 48 determines whether any performance value has reached an alertable point when compared to a predetermined threshold. If not, then the process continues in step 120 with the interval timer being reset. If a value has reached an alertable level, then in step 128 alert thread 48 generates an alert event and sends it to the local event log maintained by operating system 64. Then, in step 130, alert thread 48 generates an appropriate message regarding the alert and places it in event queue 52. To avoid sending alerts out twice, log capture thread 46 automatically filters out any alert events that were placed in the local event log of operating system 64.

FIG. 8 illustrates a flow chart of the operation of an embodiment of logging thread 50. The process begins at step 132 with initialization of logging thread 50. In step 134, logging thread 50 sets an interval timer to a predetermined interval as defined by the user of monitoring and tracking agent 16. This interval may or may not be the same interval used by alert thread 48. Next, in step 136, logging thread 50 waits for the expiration of the interval timer. When the interval timer expires, logging thread 50 retrieves performance counters from operating system 64 and computes performance values based upon those performance counters in step 138. Then, in step 140, the performance values that were computed and any performance counters that the user of monitoring and tracking agent 16 is configured to record are stored in a local log file in step 142. Next, in step 142, a message that the data has been logged along with date and time stamp information is sent to the connection list 60 by performance data procedure 58. If a previous date and time stamp remains in connection list 60, then, in this embodiment, the previous date and time stamp remains unchanged and the listener is not notified a second time. Alternatively, the listener could be notified multiple times. If no previous date and time stamp is present, then the listener is notified that performance data is available by connection manager thread 62. The process then repeats itself by resetting the interval timer in step 134.

FIG. 9 illustrates a flow chart generally describing the operation of an embodiment of monitoring and tracking listener 18. The process begins in step 144 with initialization. Then, in step 146, event queue thread 74 waits for a message to appear in event queue file 72. The messages placed in event queue file 72 are generated by event procedure 68. When a message appears, the process continues in step 148 as event queue thread 74 retrieves the message from event queue file 72. In step 150, event queue thread 74 processes the message as necessary. Then, in step 152, event queue thread 74 sends the data to the listener extensions 20. In this embodiment, event queue thread 74 sends the data to ASCII extension agent 80, SQL database extension agent 82, and console extension agent 84. Event queue thread 74 may maintain a queue of data to be sent to the listener extensions 20. Event queue thread 74 may maintain a separate queue for each listener extension 20 or a single queue for all listener extensions 20. After the data for a message has been sent to the proper listener extension, the process continues in step 146 where event queue thread 74 waits for another message to appear in event queue file 72. The process continues in this manner until the execution of monitoring and tracking listener 18 is terminated.

FIG. 10 illustrates a flow chart describing the operation of an embodiment of statistics gathering thread 76. Statistics gathering thread 76 monitors agent list 78 and, in this example, performs the process illustrated in FIG. 10 only when monitoring and tracking listener 18 does not have configuration data, events, or alerts to process. The process begins at step 154 with initialization of statistics gathering thread 76. In step 156, statistics gathering thread 76 accesses agent list 78 to determine whether any statistics are available from any monitoring and tracking agent 16. In step 158, statistics gathering thread 76 determines if such statistics are available. If not, then statistics gathering thread 76 continues to monitor agent list 78 in step 156. When statistics are available, the process continues in step 160 where statistics gathering thread 76 makes a remote procedure call to the appropriate agent and obtains all performance statistics that the monitoring and tracking agent 16 has recorded since the date and time entry for that agent in agent list 78. After obtaining this data, the statistics that were obtained from the remote procedure call are packaged into a message and sent to the listener extensions 20 in step 162. These messages may be queued by the listener extensions 20. The process then continues in step 156.

The processes illustrated in FIGS. 5–10 are only examples illustrating the operation of monitoring and tracking agent 16 and monitoring and tracking listener 18. Other processes could be used without departing from the scope of the invention. The functions of monitoring and tracking 16 and monitoring and tracking listener 18 could also be performed by a computer software program having a different structure than the embodiments illustrated in FIGS. 3 and 4. Various steps could also be omitted or additional steps added to the processes illustrated in FIGS. 5 through 10.

Although the present invention has been described in detail, it should be understood that various changes,

substitutions, and alterations can be made hereto without departing from the spirit and scope of the invention as defined by the appended claims.

What is claimed is:

1. A method for monitoring the performance of a first computer coupled to a computer network, comprising:

repeatedly obtaining performance data comprising at least one performance value using the first computer, the performance value comprising a measure of the performance of the first computer;

automatically sending the performance data from the first computer over the computer network to a second computer coupled to the computer network;

receiving the performance data at the second computer; and

passing the performance data to a first extension agent, the first extension agent operable to process the performance data.

2. The method of claim 1 wherein the first extension agent comprises a dynamic link library procedure running on the second computer.

3. The method of claim 1 wherein the first extension agent is further operable to send the processed performance data to a database.

4. The method of claim 1 wherein the first extension agent is further operable to send the processed performance data to a central monitoring console.

5. The method of claim 1, further comprising:

queueing the performance data received by the second computer; and

wherein the passing step further comprises passing the queued performance data to the first extension agent when the first extension agent is ready to receive the performance data.

6. The method of claim 1, further comprising:

passing the performance data to a second extension agent, the second extension agent operable to process the performance data.

7. The method of claim 6, further comprising:

queueing the performance data received by the second computer to a first queue and a second queue;

wherein the step of passing the performance data to the first extension agent further comprises passing the queued performance data in the first queue to the first extension agent when the first extension agent is ready to receive the performance data; and

wherein the step of passing the performance data to the second extension agent further comprises passing the queued performance data in the second queue to the second extension agent when the second extension agent is ready to receive the performance data.

8. A system for monitoring the performance of a first computer coupled to a computer network, comprising:

a first computer;

a second computer;

a computer network connecting the first and second computer;

a tracking program running on the first computer and operable to repeatedly obtain performance data comprising at least one performance value, the performance value comprising a measure of the performance of the first computer, the tracking program further operable to automatically send the performance data from the first computer over the computer network to the second computer; and

a dispatching program running on the second computer, the dispatching program operable to receive the performance data and pass the performance data to a first extension agent, the first extension agent operable to process the performance data.

9. The system of claim 8, wherein the first extension agent comprises a dynamic link library procedure running on the second computer.

10. The system of claim 8, wherein the first extension agent is further operable to send the processed performance data to a database.

11. The system of claim 8, wherein the first extension agent is further operable to send the processed performance data to a central monitoring console.

12. The system of claim 11, wherein the first extension agent is further operable to queue events contained in the processed performance data to a central monitoring console.

13. The system of claim 8, wherein the tracking program is further operable to obtain configuration data comprising information about the configuration of the first computer and automatically send the configuration data over the computer network to the second computer.

14. The system of claim 8, wherein the tracking program is further operable to generate an alert indicating that the performance of the first computer has reached a critical level and automatically send the alert over the computer network to the second computer.

15. The system of claim 8, wherein the tracking program is further operable to monitor events that occur in the first computer and are recorded by an operating system controlling the first computer, the tracking program further operable to send the events over the computer network to the second computer.

16. The system of claim 8, wherein the dispatching program is further operable to queue the performance data received by the second computer and pass the queued performance data to the first extension agent when the first extension agent is ready to receive the performance data.

17. The system of claim 8, wherein the dispatching program is further operable to pass the performance data to a second extension agent running on the second computer, the second extension agent operable to process the performance data.

18. The system of claim 17, wherein the dispatching program is further operable to queue the performance data received by the second computer to a first queue and a second queue, pass the queued performance data in the first queue to the first extension agent when the first extension agent is ready to receive the performance data, and pass the queued performance data in the second queue to the second extension agent when the second extension agent is ready to receive the performance data.

19. A program for monitoring the performance of a first computer coupled to a computer network, comprising:

a computer readable storage medium;

a dispatching program stored on the storage medium, the dispatching program operable to receive performance data sent over the computer network by the first computer, the performance data repeatedly obtained by the first computer and comprising a measure of the performance of the first computer, the dispatching program further operable to pass the performance data to a first extension agent, the first extension agent operable to process the performance data.

20. The program of claim 19, wherein the dispatching program is further operable to queue the performance data received by the second computer and pass the queued performance data to the first extension agent when the first extension agent is ready to receive the performance data.

* * * * *

US006373498B1

(12) **United States Patent**
Abgrall

(10) **Patent No.: US 6,373,498 B1**
(45) **Date of Patent: Apr. 16, 2002**

(54) **DISPLAYING IMAGES DURING BOOT-UP AND SHUTDOWN**

(75) Inventor: **Jean-Paul Abgrall**, San Jose, CA (US)

(73) Assignee: **Phoenix Technologies Ltd.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/336,003**

(22) Filed: **Jun. 18, 1999**

(51) **Int. Cl.**[7] **G09G 5/34**
(52) **U.S. Cl.** **345/619**
(58) **Field of Search** 345/133, 24, 592, 345/619, 765, 778, 811; 711/205, 207; 713/1, 2; 714/36

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,121,345	A		6/1992	Lentz
5,128,995	A		7/1992	Arnold et al.
5,131,089	A		7/1992	Cole
5,142,680	A		8/1992	Ottman et al.
5,146,568	A		9/1992	Flaherty et al.
5,214,695	A		5/1993	Arnold et al.
5,274,816	A		12/1993	Oka
5,280,627	A		1/1994	Flaherty et al.
5,307,497	A		4/1994	Feigenbaum et al.
5,325,532	A		6/1994	Crosswy et al.
5,379,431	A		1/1995	Lemon et al.
5,381,549	A		1/1995	Tamura
5,418,918	A		5/1995	Vander Kamp et al.
5,444,850	A		8/1995	Chang
5,448,741	A		9/1995	Oka
5,452,454	A		9/1995	Basu
5,463,766	A		10/1995	Schieve et al.
5,469,573	A		11/1995	McGill, III et al.
5,504,905	A		4/1996	Cleary et al.
5,522,076	A		5/1996	Dewa et al.
5,526,523	A		6/1996	Straub et al.
5,542,082	A		7/1996	Solhjell
5,581,740	A		12/1996	Jones
5,586,327	A		12/1996	Bealkowski et al.
5,594,903	A		1/1997	Bunnell et al.
5,604,890	A		2/1997	Miller
5,652,868	A		7/1997	Williams
5,652,886	A		7/1997	Tulpule et al.
5,664,194	A		9/1997	Paulsen
5,680,547	A		10/1997	Chang
5,692,190	A		11/1997	Williams
5,694,583	A		12/1997	Williams et al.
5,694,600	A		12/1997	Khenson et al.
5,701,477	A		12/1997	Chejlava, Jr.
5,715,456	A		2/1998	Bennett et al.
5,717,930	A		2/1998	Imai et al.
5,727,213	A		3/1998	Vander Kamp et al.
5,732,268	A		3/1998	Bizzarri
5,748,957	A		5/1998	Klein
5,754,853	A		5/1998	Pearce
5,764,593	A		6/1998	Turpin et al.
5,781,758	A		7/1998	Morley
5,790,849	A		8/1998	Crocker et al.
5,796,984	A		8/1998	Pearce et al.
5,802,363	A		9/1998	Williams et al.
5,805,880	A		9/1998	Pearce et al.
5,805,882	A		9/1998	Cooper et al.
5,815,706	A		9/1998	Stewart et al.
5,819,063	A		10/1998	Dahl et al.
5,828,888	A		10/1998	Kozaki et al.
5,832,251	A		10/1998	Takahashi
5,842,011	A		11/1998	Basu
5,854,905	A		12/1998	Garney
5,864,698	A		1/1999	Krau et al.
5,887,164	A		3/1999	Gupta
5,901,310	A		5/1999	Rahman et al.
5,907,679	A		5/1999	Hoang et al.
6,049,871	A	*	4/2000	Silen et al. 713/2
6,205,450	B1	*	3/2001	Kanome 707/203

* cited by examiner

Primary Examiner—Almis R. Jankus
Assistant Examiner—J. F. Cunningham

(57) **ABSTRACT**

The present invention is a method and apparatus to display an image during a transition of an operating system in a computer system. An image having an image format compatible with the operating system is obtained. Content of a system file corresponding to the transition of the operating system is created using the image in a system directory.

45 Claims, 8 Drawing Sheets

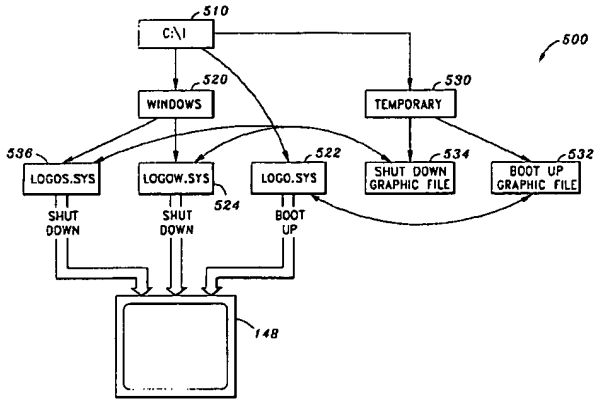
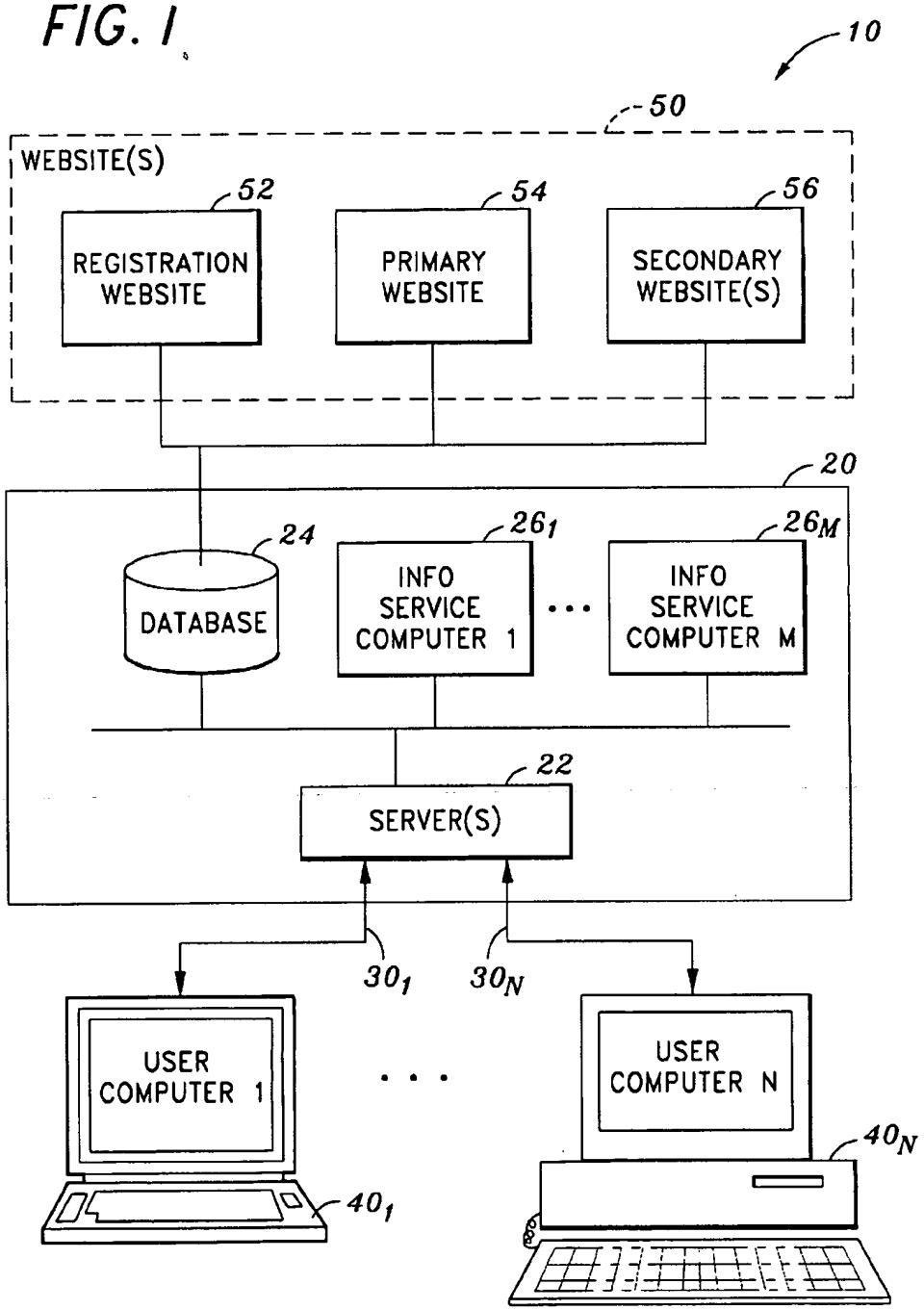


FIG. 1


10
50
WEBSITE(S)
52
REGISTRATION WEBSITE
54
PRIMARY WEBSITE
56
SECONDARY WEBSITE(S)
20
24
DATABASE
26₁
INFO SERVICE COMPUTER 1
26M
INFO SERVICE COMPUTER M
22
SERVER(S)
30₁
30N
USER COMPUTER 1
USER COMPUTER N
40₁
40N

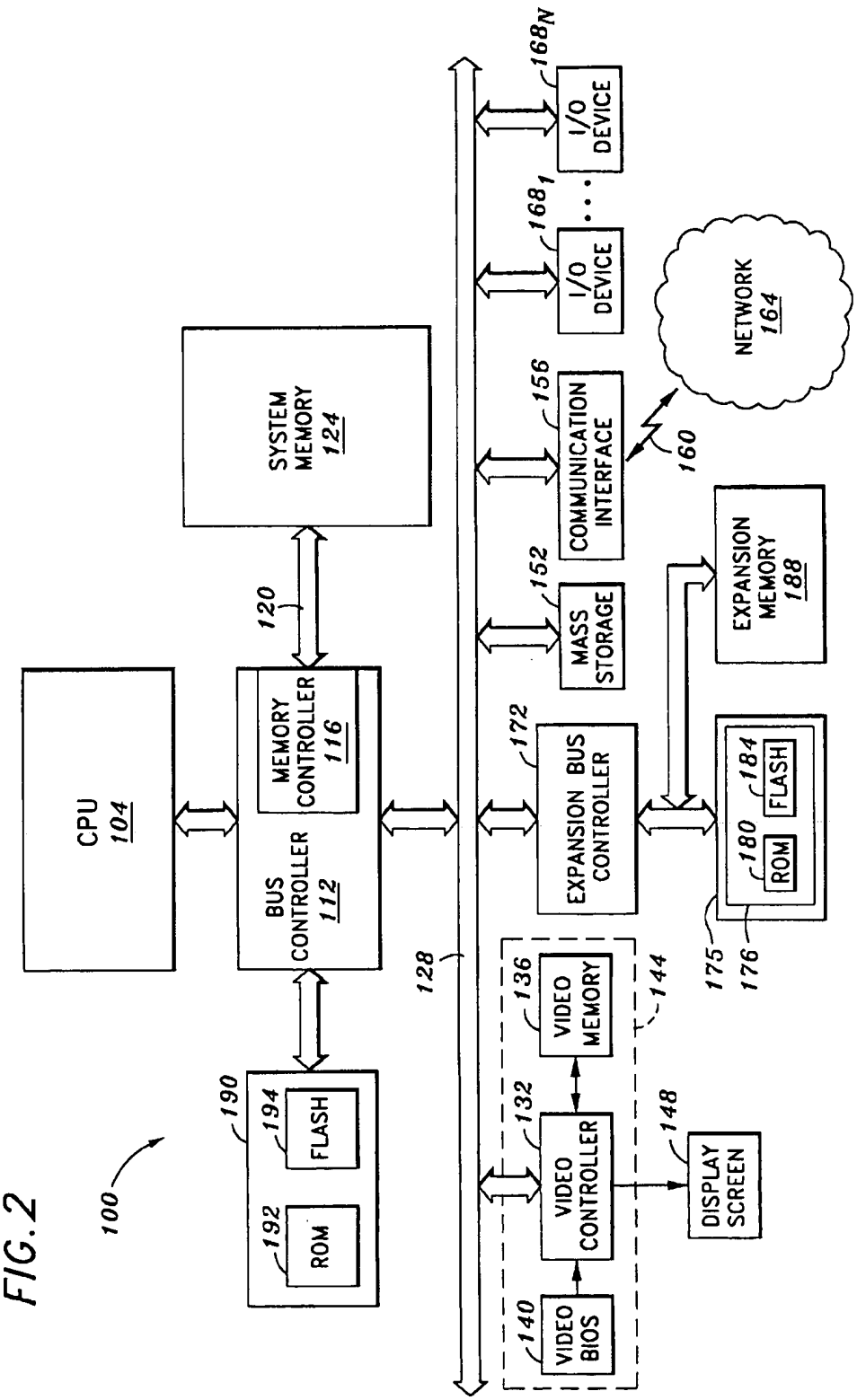
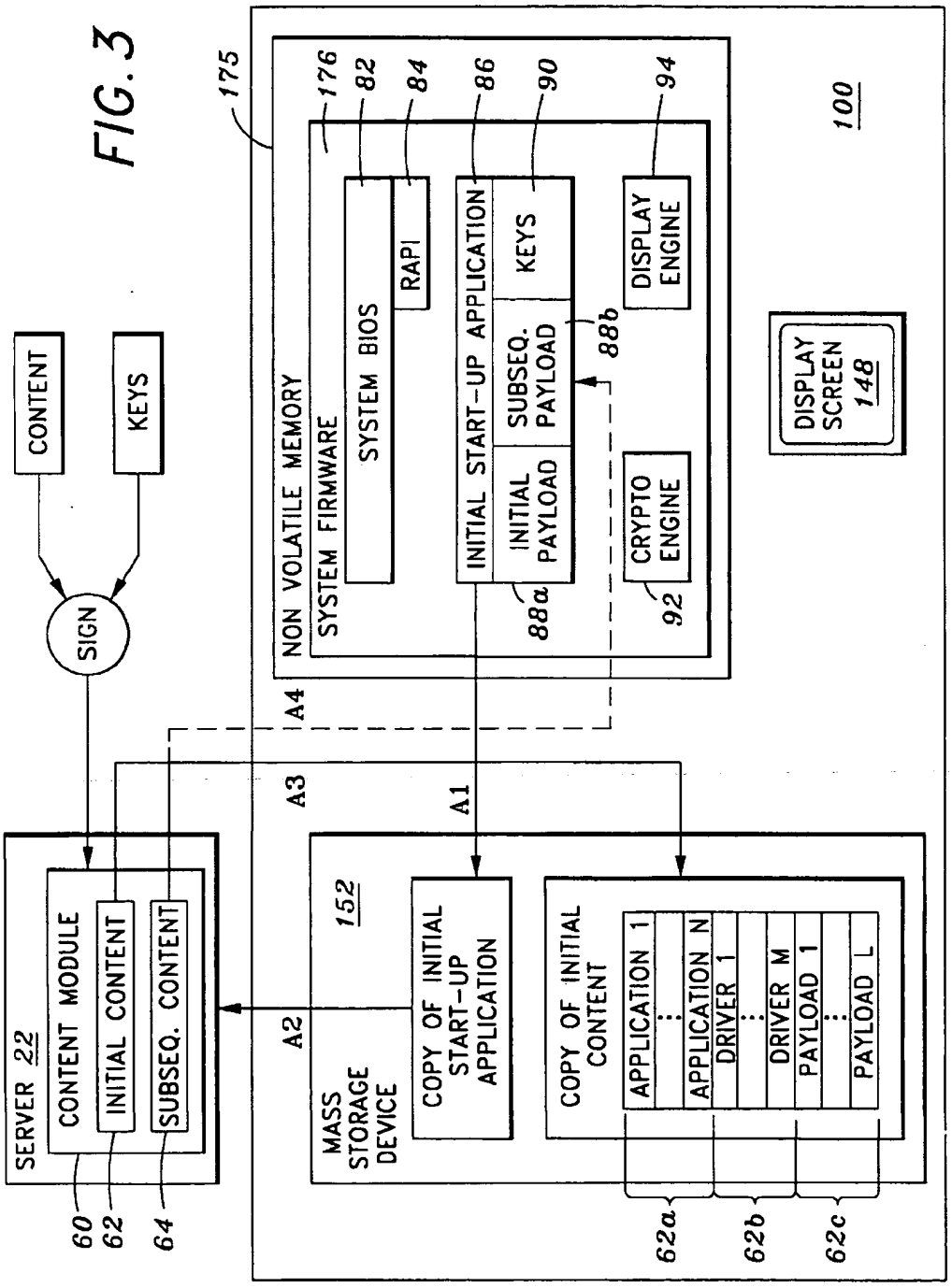

100
CPU 104
BUS CONTROLLER 112
MEMORY CONTROLLER 116
120
SYSTEM MEMORY 124
128
190
194
FLASH
192
ROM
132
VIDEO CONTROLLER
136
VIDEO MEMORY
144
140
VIDEO BIOS
148
DISPLAY SCREEN
172
EXPANSION BUS CONTROLLER
152
MASS STORAGE
156
COMMUNICATION INTERFACE
160
NETWORK 164
168N
I/O DEVICE
1681
I/O DEVICE
EXPANSION MEMORY 188
184
FLASH
180
ROM
175
176

FIG. 2

FIG. 3


100
175
NON VOLATILE MEMORY
176
SYSTEM FIRMWARE
82
SYSTEM BIOS
84
RAPI
86
INITIAL START-UP APPLICATION
88a
INITIAL PAYLOAD
90
SUBSEQ. PAYLOAD
88b
KEYS
92
CRYPTO ENGINE
94
DISPLAY ENGINE
DISPLAY SCREEN 148
CONTENT
KEYS
SIGN
A4
A3
A1
A2
SERVER 22
CONTENT MODULE
INITIAL CONTENT
SUBSEQ. CONTENT
60
62
64
152
MASS STORAGE DEVICE
COPY OF INITIAL START-UP APPLICATION
COPY OF INITIAL CONTENT
APPLICATION 1
APPLICATION N
DRIVER 1
DRIVER M
PAYLOAD 1
PAYLOAD L
62a
62b
62c

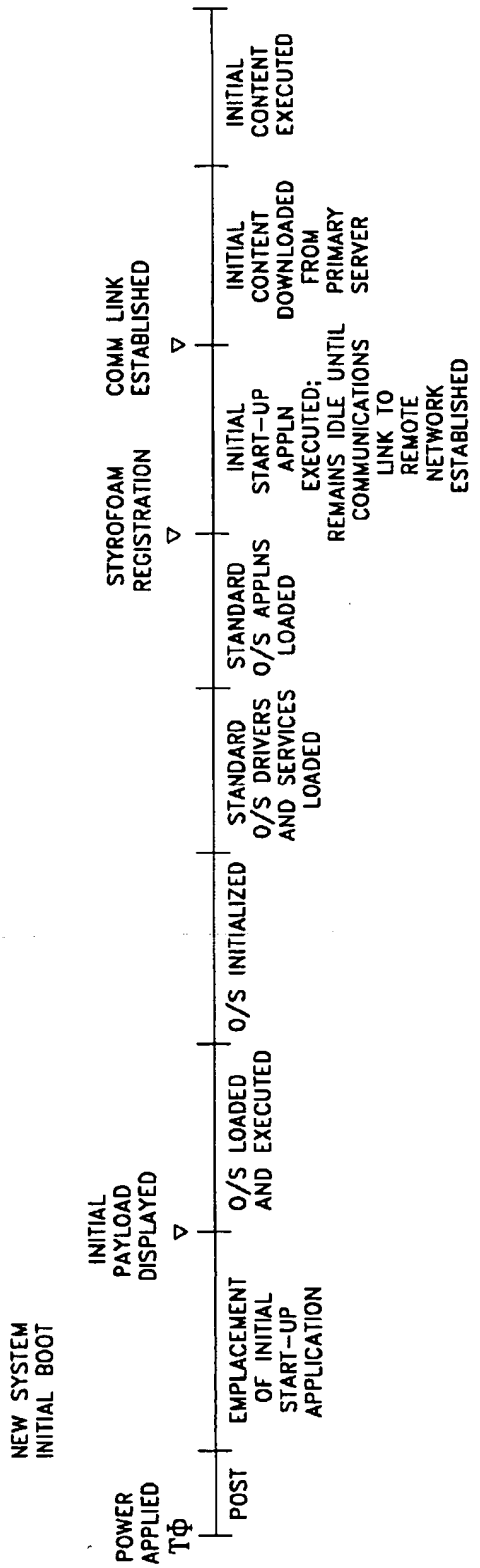

NEW SYSTEM INITIAL BOOT
POWER APPLIED TΦ
POST
EMPLACEMENT OF INITIAL START-UP APPLICATION
INITIAL PAYLOAD DISPLAYED
O/S LOADED AND EXECUTED
O/S INITIALIZED
STANDARD O/S DRIVERS AND SERVICES LOADED
STYROFOAM REGISTRATION
STANDARD O/S APPLNS LOADED
INITIAL START-UP APPLN EXECUTED; REMAINS IDLE UNTIL COMMUNICATIONS LINK TO REMOTE NETWORK ESTABLISHED
COMM LINK ESTABLISHED
INITIAL CONTENT DOWNLOADED FROM PRIMARY SERVER
INITIAL CONTENT EXECUTED
INFOMEDIARY DRIVERS INSTALLED
INFOMEDIARY APPLNS INSTALLED
APPLNS AND DRIVERS EXECUTED; PROFILE MGR SENDS USER AND HARDWARE PROFILE TO SERVER CONTINUOUSLY UNTIL POWER DOWN
PAYLOAD 1-N DOWNLOADED
MEMORY UPDATED
END

FIG. 4A

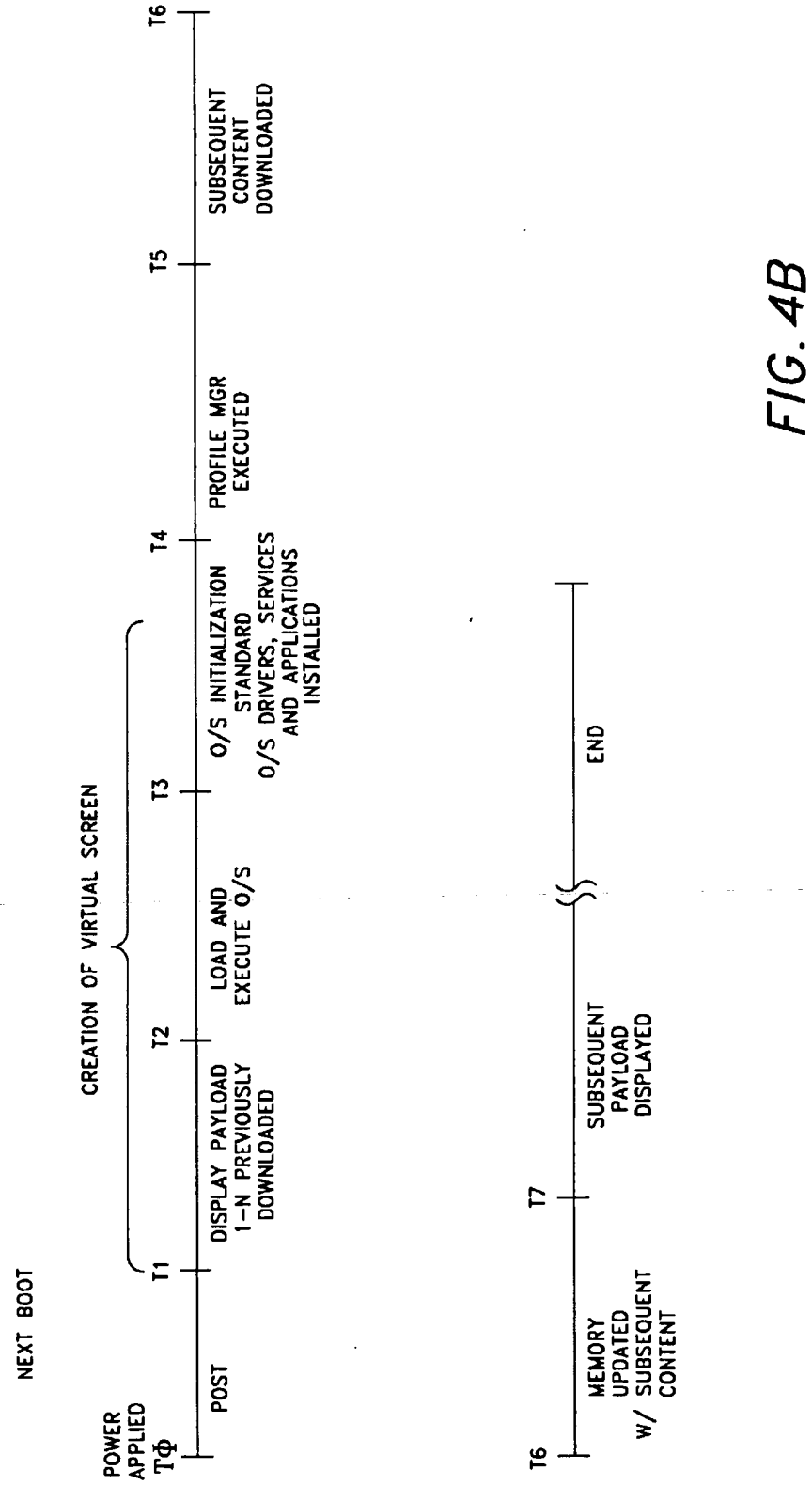

POWER APPLIED
TΦ
POST
T1
DISPLAY PAYLOAD 1-N PREVIOUSLY DOWNLOADED
T2
LOAD AND EXECUTE O/S
T3
O/S INITIALIZATION STANDARD O/S DRIVERS, SERVICES AND APPLICATIONS INSTALLED
T4
PROFILE MGR EXECUTED
T5
SUBSEQUENT CONTENT DOWNLOADED
T6
CREATION OF VIRTUAL SCREEN
NEXT BOOT
T6
MEMORY UPDATED W/ SUBSEQUENT CONTENT
T7
SUBSEQUENT PAYLOAD DISPLAYED
END

FIG. 4B

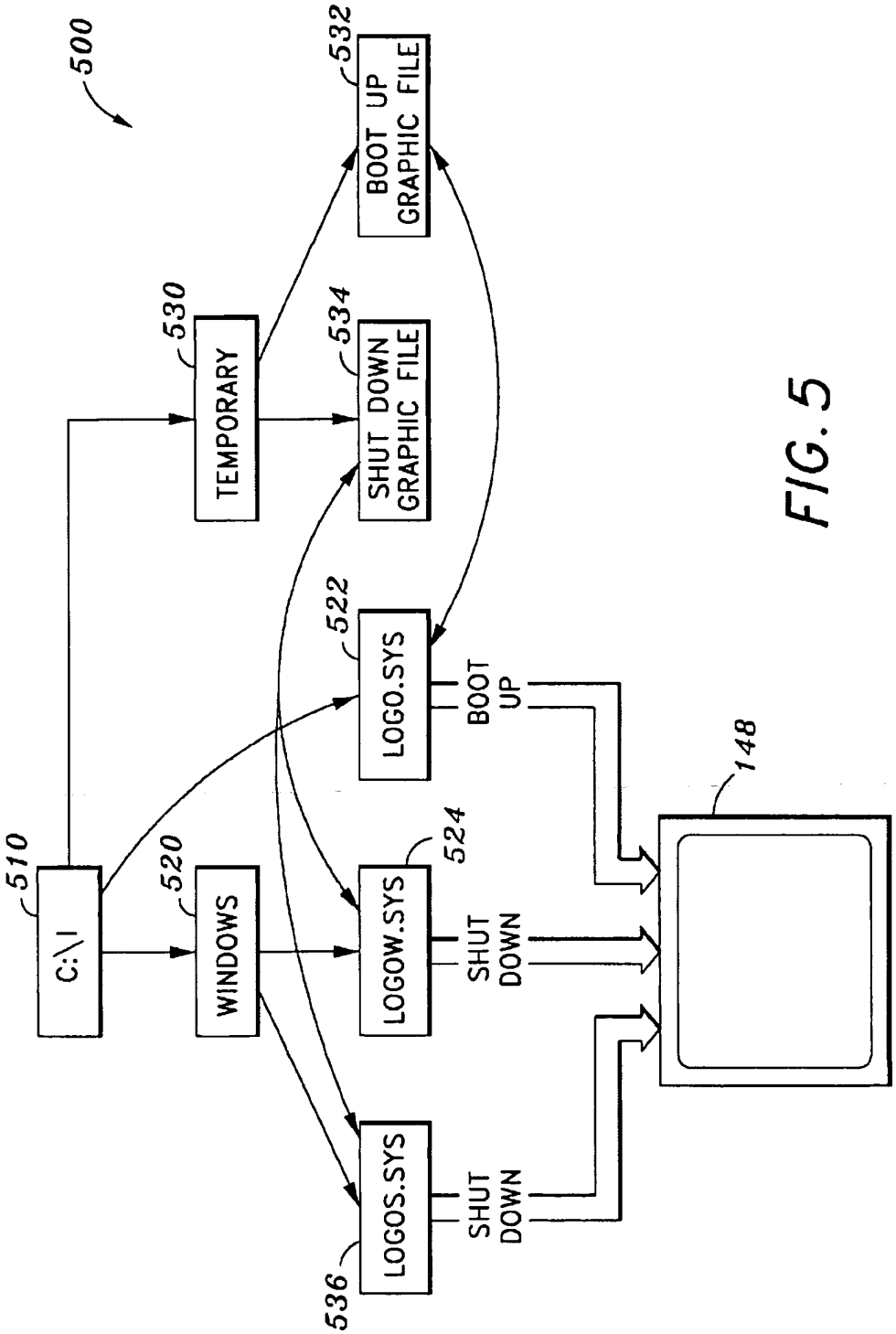

500
532
BOOT UP GRAPHIC FILE
530
TEMPORARY
534
SHUT DOWN GRAPHIC FILE
522
LOGO.SYS
BOOT UP
148
510
C:\
520
WINDOWS
524
LOGOW.SYS
SHUT DOWN
LOGOS.SYS
SHUT DOWN
536

FIG. 5

FIG. 6

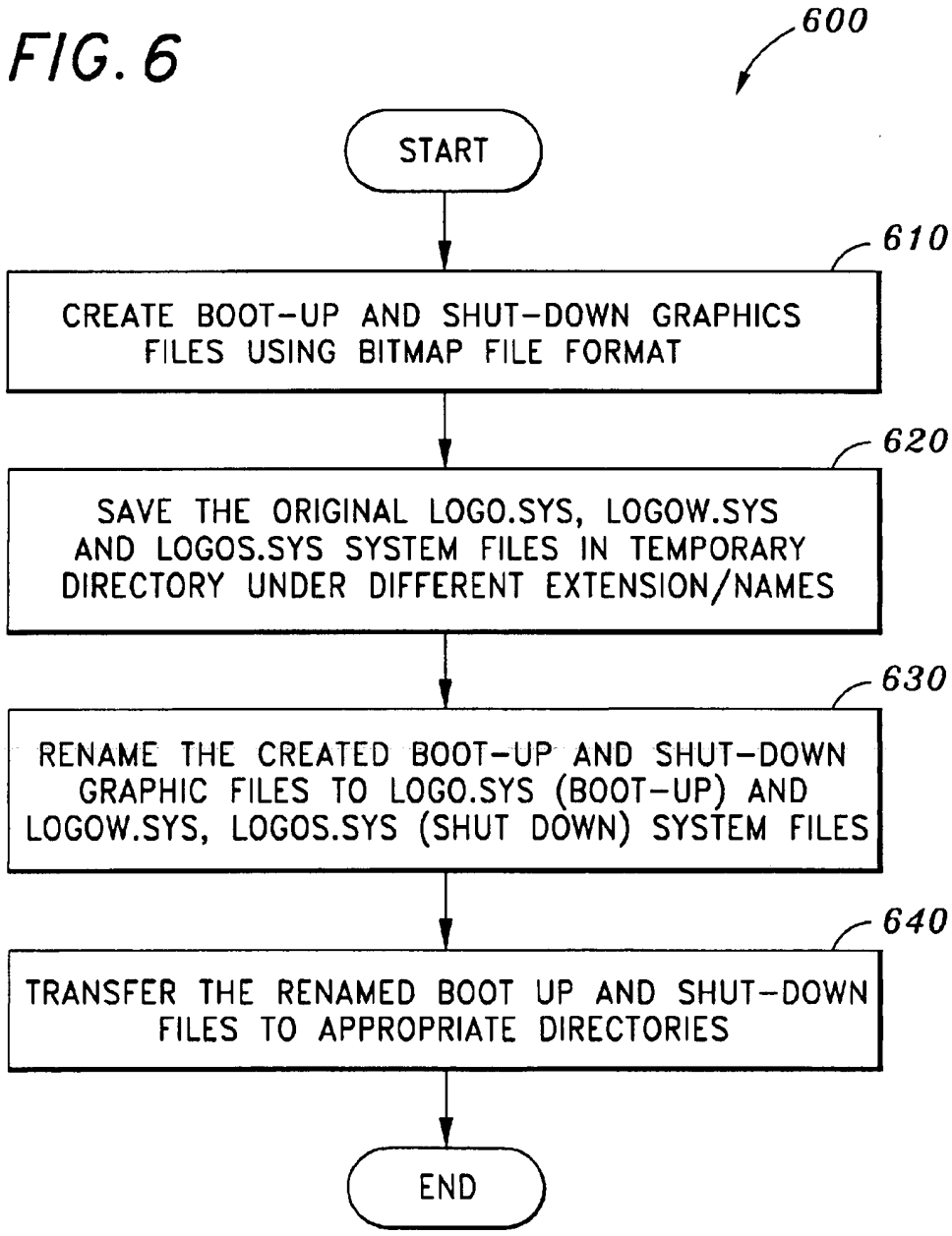
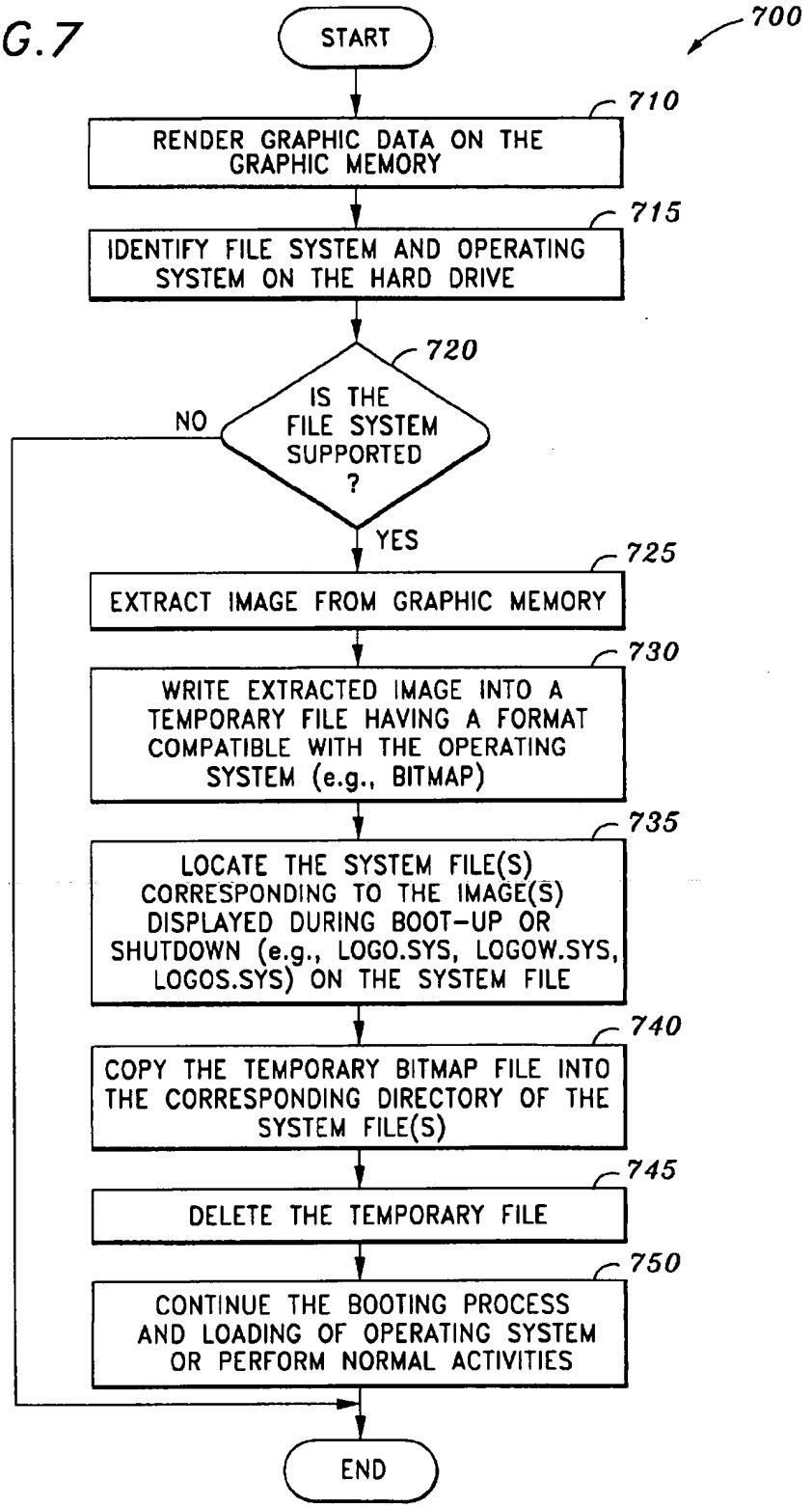

600
START
610
CREATE BOOT-UP AND SHUT-DOWN GRAPHICS FILES USING BITMAP FILE FORMAT
620
SAVE THE ORIGINAL LOGO.SYS, LOGOW.SYS AND LOGOS.SYS SYSTEM FILES IN TEMPORARY DIRECTORY UNDER DIFFERENT EXTENSION/NAMES
630
RENAME THE CREATED BOOT-UP AND SHUT-DOWN GRAPHIC FILES TO LOGO.SYS (BOOT-UP) AND LOGOW.SYS, LOGOS.SYS (SHUT DOWN) SYSTEM FILES
640
TRANSFER THE RENAMED BOOT UP AND SHUT-DOWN FILES TO APPROPRIATE DIRECTORIES
END

FIG. 7


700
START
710
RENDER GRAPHIC DATA ON THE GRAPHIC MEMORY
715
IDENTIFY FILE SYSTEM AND OPERATING SYSTEM ON THE HARD DRIVE
720
IS THE FILE SYSTEM SUPPORTED ?
NO
YES
725
EXTRACT IMAGE FROM GRAPHIC MEMORY
730
WRITE EXTRACTED IMAGE INTO A TEMPORARY FILE HAVING A FORMAT COMPATIBLE WITH THE OPERATING SYSTEM (e.g., BITMAP)
735
LOCATE THE SYSTEM FILE(S) CORRESPONDING TO THE IMAGE(S) DISPLAYED DURING BOOT-UP OR SHUTDOWN (e.g., LOGO.SYS, LOGOW.SYS, LOGOS.SYS) ON THE SYSTEM FILE
740
COPY THE TEMPORARY BITMAP FILE INTO THE CORRESPONDING DIRECTORY OF THE SYSTEM FILE(S)
745
DELETE THE TEMPORARY FILE
750
CONTINUE THE BOOTING PROCESS AND LOADING OF OPERATING SYSTEM OR PERFORM NORMAL ACTIVITIES
END

DISPLAYING IMAGES DURING BOOT-UP AND SHUTDOWN

BACKGROUND

1. Field of the Invention

This invention relates to graphics. In particular, the invention relates to graphic display.

2. Description of Related Art

A typical process of loading an operating system (OS) from a basic input and output system (BIOS) takes some time to complete. During this time, typically the display screen displays an image as selected by the operating system. This image is fixed and is not changed by the OS. Similarly, when the system is shutdown, the OS goes through a shutdown sequence and displays images on the screen during the shutdown process. The boot-up and shutdown images as displayed by the OS are normally not useful to the user and merely contain routine messages.

Since the time to boot up and shut down is sufficiently long for the system to display more informative images, it is desirable to be able to display images other than the standard logos of the operating system.

Therefore there is a need in the technology to provide a simple and efficient method to display an image during a transition of the operating system.

SUMMARY

The present invention is a method and apparatus to display an image during a transition of an operating system in a computer system. An image having an image format compatible with the operating system is obtained. Content of a system file corresponding to the transition of the operating system is created using the image in a system directory.

BRIEF DESCRIPTION OF THE DRAWINGS

The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

FIG. **1** is a system block diagram of one embodiment of an information distribution system in which the apparatus and method of the invention is used.

FIG. **2** illustrates an exemplary processor system or user computer system which implements embodiments of the present invention.

FIG. **3** illustrates a logical diagram of one embodiment of the invention.

FIG. 4A and FIG. 4B illustrates one embodiment of a process flow chart provided in accordance with the principles of the invention.

FIG. **5** is a diagram illustrating an architecture to display an image during a transition of the operating system according to one embodiment of the invention.

FIG. **6** is a flowchart illustrating a process to display an image during a transition of the operating system according to one embodiment of the invention.

FIG. **7** is a flowchart illustrating a process to display an image during a transition of the operating system according to another embodiment of the invention.

DESCRIPTION

The present invention is a method and apparatus to display an image during a transition of an operating system such as boot-up and shutdown. A boot-up graphic file replaces a boot-up system file in a system directory. A shutdown graphic file replaces a shutdown system file in a system directory. The technique allows the display of images other than the standard images by the operating system.

In the following description, for purposes of explanation, numerous details are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that these specific details are not required in order to practice the present invention. In other instances, well known electrical structures and circuits are shown in block diagram form in order not to obscure the present invention.

Definitions

As discussed herein, a "computer system" is a product including circuitry capable of processing data. The computer system may include, but is not limited to, general purpose computer systems (e.g., server, laptop, desktop, palmtop, personal electronic devices, etc.), personal computers (PCs), hard copy equipment (e.g., printer, plotter, fax machine, etc.), banking equipment (e.g., an automated teller machine), and the like. An infomediary is a web site that provides information on behalf of producers of goods and services, supplying relevant information to businesses about products and/or services offered by suppliers and other businesses. Content refers to application programs, driver programs, utility programs, the payload, etc., and combinations thereof, as well as graphics, informational material (articles, stock quotes, etc.) and the like, either singly or in any combination. "Payload" refers to messages with graphics or informational material (such as, articles, stock quotes, etc.) and may include files or applications. In one embodiment, it is transferred at a predetermined time to the system's mass storage media. In addition, a "communication link" refers to the medium or channel of communication. The communication link may include, but is not limited to, a telephone line, a modem connection, an Internet connection, an Integrated Services Digital Network ("ISDN") connection, an Asynchronous Transfer Mode (ATM) connection, a frame relay connection, an Ethernet connection, a coaxial connection, a fiber optic connection, satellite connections (e.g. Digital Satellite Services, etc.), wireless connections, radio frequency (RF) links, electromagnetic links, two way paging connections, etc., and combinations thereof.

In addition, the loading of an operating system ("OS") refers to the initial placement of the operating system bootstrap loader. In one embodiment, during the OS load, a sector of information is typically loaded from a hard disk into the system memory. Alternatively, the bootstrap loader is loaded from a network into system memory. An OS "boot" refers to the execution of the bootstrap loader. This places the OS in control of the system. Some of the actions performed during the OS boot include system configuration, device detection, loading of drivers and user logins. OS runtime refers to the completion of the boot phase and the beginning of the execution of applications by the OS. In one embodiment, during OS runtime, the OS interacts with the user to execute and/or run applications.

Power On Self Test (POST) refers to the instructions that are executed to configure and test the system hardware prior to loading an OS.

System Overview

A description of an exemplary system, which incorporates embodiments of the present invention, is hereinafter described.

FIG. 1 shows a system block diagram of one embodiment of an information distribution system 10 in which the apparatus and method of the invention is used. The system 10 relates to providing an infomediary. It involves the construction and maintenance of a secure and private repository of Internet user and system profiles, collected primarily from warranty service registrations, Internet service registrations, system profiles, and user preferences. Initially, this information is used to register the user with the manufacturers of purchased hardware and software products, and with the providers of on-line or other services. Over time, the user data is used to create a user profile and notify users of relevant software updates and upgrades, to encourage on-line purchases of related products, and to enable one-to-one customized marketing and other services.

In one embodiment, two software modules are used to implement various embodiments of the invention. One is resident on a user's system, and is used to access a predetermined web site. For example, in one embodiment, the operating system and Basic Input and Output System (BIOS) are pre-installed on a computer system, and when the computer system is subsequently first powered up, an application, referred to for discussion purposes as the first software module (in one embodiment, the first software module is the initial start-up application (ISUA), which will be described in the following sections), will allow the launching of one or more executable programs in the pre-boot environment. In one embodiment, the first software module facilitates the launching of one or more executable programs prior to the loading, booting, execution and/or running of the OS. In one embodiment, the user is encouraged to select the use of such a program (i.e., the use of the first software module), and in alternative embodiments, the program is automatically launched. The program(s) contained in the first software module enables tools and utilities to run at an appropriate time, and with proper user authorization, also allow the user to download a second software module that includes drivers, applications and additional payloads through the Internet connection on the PC. The programs may also provide for remote management of the system if the OS fails to launch successfully.

Once the second software module has been delivered, it may become memory resident, and may disable the transferred copy of the first software module. The original copy of the first software module still residing in the system's non-volatile memory remains idle until the second software module fails to function, becomes corrupted or is deleted, upon which a copy of the original first software module is again transferred as described above. The second software module may include an application that connects the user to a specific server on the Internet and directs the user to a predetermined web site to seek authorization to down load further subscription material. The second software module may also include content that is the same or similar to the content of the first software module.

In one embodiment, the system may also include an initial payload that is stored in Read Only Memory BIOS (ROM BIOS). In one embodiment, the initial payload is part of the first software module (e.g., the ISUA). In an alternative embodiment, the initial payload is stored as a module in ROM BIOS, separate from the first software module. In one embodiment, the initial payload is launched from ROM BIOS and displayed on the screen after the Power On Self Test (POST) but prior to the booting, loading and/or execution of the OS. This may occur at a predetermined time, such as when the system is being manufactured, assembled and tested, or when the end user first activates the system. In an alternate embodiment, this initial payload is copied to a predetermined location (such as the system's hard disk) at a predetermined time, such as when the system is being manufactured, assembled and tested, or when the end user first activates the system. Once copied, the payload executes after POST but prior to operation of the OS, and may display graphics, advertisements, animation, Joint Photographic Experts Group (JPEG)/Moving Picture Experts Group (MPEG) formatted material on the screen. When additional programs and/or payloads are delivered (via the Internet or other outside connection), the display screen may be used to provide customized screens in the form of messages or graphics prior to and during booting of the OS. In addition, executable programs delivered in the first software module, as well as subsequent programs (such as the second software module) downloaded from the web site, may be used to survey the PC to determine various types of devices, drivers, and applications installed. In one embodiment, as described in co-pending U.S. patent application Ser. No 09/336,289 , entitled "Method and Apparatus for Automatically Installing And Configuring Software on a Computer" incorporated herein by reference, the first software module is used to identify and to automatically create shortcuts and/or bookmarks for the user. The programs downloaded from the website may include software that collects and maintains a user profile based on the user's preferences. Such information may be provided to the infomediary, which subsequently forwards portions of the information and/or compiled data based on the information to suppliers and other businesses to obtain updates or revisions of information provided by the suppliers and other businesses.

Referring to FIG. 1, the information distribution system 10 comprises a service center 20 that is connected over one or more communications links 30_1–30_N to one or more user computer systems 40_1–40_N ("40"). The service center 20 includes one or more servers 22, one or more databases 24, and one or more computers 26_1–26_M. The one or more computers 26_1–26_M are capable of simultaneous access by a plurality of the user computer systems 40_1–40_N. If a plurality of computers are used, then the computers 26_1–26_M may be connected by a local area network (LAN) or any other similar connection technology. However, it is also possible for the service center 20 to have other configurations. For example, a smaller number of larger computers (i.e. a few mainframe, mini, etc. computers) with a number of internal programs or processes running on the larger computers capable of establishing communications links to the user computers.

The service center 20 may also be connected to a remote network 50 (e.g., the Internet) or a remote site (e.g., a satellite, which is not shown in FIG. 1). The remote network 50 or remote site allows the service center 20 to provide a wider variety of computer software, content, etc. that could be stored at the service center 20. The one or more databases 24 connected to the service center computer(s), e.g., computer 26_1, are used to store database entries consisting of computer software available on the computer(s) 26. In one embodiment, each user computer 40_1–40_N has its own secure database (not shown), that is not accessible by any other computer. The communication links 30_1–30_N allow the one or more user computer systems 40_1–40_N to simultaneously connect to the computer(s) 26_1–26_M. The connections are managed by the server 22.

After a user computer system 40 establishes two-way communications with the information service computer 26, the content is sent to the user computer system 40 in a manner hereinafter described. The downloaded content

includes an application that surveys the user and/or the user computer system's hardware and/or software to develop a user profile as well as a profile of the user's system. The information gathered from the user and/or user's computer system is subsequently provided to the service center **20**, which provides additional content to the user computer **40** based on the user and system profile. The database entries from the database connected to the service computer **26** contain information about computer software, hardware, and third party services and products that are available to a user. Based on the user and/or system profile, the content is further sent to the user computer for display. The content may also include a summary of information such as the availability of patches and fixes for existing computer software, new versions of existing computer software, brand new computer software, new help files, etc. The content may further include information regarding availability of hardware and third party products and services that is of interest to the user. The user is then able to make one or more choices from the summary of available products and services, and request that the products be transferred from the service computer **26** to the user computer. Alternatively, the user may purchase the desired product or service from the summary of available products and services.

FIG. **2** illustrates an exemplary computer system **100** that implements embodiments of the present invention. The computer system **100** illustrates one embodiment of user computer systems 40_1–40_N and/or computers 26_1–26_M (FIG. **1**), although other embodiments may be readily used.

Referring to FIG. **2**, the computer system **100** comprises a processor or a central processing unit (CPU) **104**. The illustrated CPU **104** includes an Arithmetic Logic Unit (ALU) for performing computations, a collection of registers for temporary storage of data and instructions, and a control unit for controlling operation for the system **100**. In one embodiment, the CPU **104** includes any one of the x86, Pentium™, Pentium II™, and Pentium Pro™ microprocessors as marketed by Intel™ Corporation, the K-6 microprocessor as marketed by AMD™, or the 6x86MX microprocessor as marketed by Cyrix™ Corp. Further examples include the Alpha™ processor as marketed by Digital Equipment Corporation™, the 680X0 processor as marketed by Motorola™; or the Power PC™ processor as marketed by IBM™. In addition, any of a variety of other processors, including those from Sun Microsystems, MIPS, IBM, Motorola, NEC, Cyrix, AMD, Nexgen and others may be used for implementing CPU **104**. The CPU **104** is not limited to microprocessor but may take on other forms such as microcontrollers, digital signal processors, reduced instruction set computers (RISC), application specific integrated circuits, and the like. Although shown with one CPU **104**, computer system **100** may alternatively include multiple processing units.

The CPU **104** is coupled to a bus controller **112** by way of a CPU bus **108**. The bus controller **112** includes a memory controller **116** integrated therein, though the memory controller **116** may be external to the bus controller **112**. The memory controller **116** provides an interface for access by the CPU **104** or other devices to system memory **124** via memory bus **120**. In one embodiment, the system memory **124** includes synchronous dynamic random access memory (SDRAM). System memory **124** may optionally include any additional or alternative high speed memory device or memory circuitry. The bus controller **112** is coupled to a system bus **128** that may be a peripheral component interconnect (PCI) bus, Industry Standard Architecture (ISA) bus, etc. Coupled to the system bus **128** are a graphics controller, a graphics engine or a video controller **132**, a mass storage device **152**, a communication interface device **156**, one or more input/output (I/O) devices 168_1–168_N, and an expansion bus controller **172**. The video controller **132** is coupled to a video memory **136** (e.g., 8 Megabytes) and video BIOS **140**, all of which may be integrated onto a single card or device, as designated by numeral **144**. The video memory **136** is used to contain display data for displaying information on the display screen **148**, and the video BIOS **140** includes code and video services for controlling the video controller **132**. In another embodiment, the video controller **132** is coupled to the CPU **104** through an Advanced Graphics Port (AGP) bus.

The mass storage device **152** includes (but is not limited to) a hard disk, floppy disk, CD-ROM, DVD-ROM, tape, high density floppy, high capacity removable media, low capacity removable media, solid state memory device, etc., and combinations thereof. The mass storage device **152** may include any other mass storage medium. The communication interface device **156** includes a network card, a modem interface, etc. for accessing network **164** via communications link **160**. The I/O devices 168_1–168_N include a keyboard, mouse, audio/sound card, printer, and the like. The I/O devices 168_1–168_N may be a disk drive, such as a compact disk drive, a digital disk drive, a tape drive, a zip drive, a jazz drive, a digital video disk (DVD) drive, a solid state memory device, a magneto-optical disk drive, a high density floppy drive, a high capacity removable media drive, a low capacity media device, and/or any combination thereof. The expansion bus controller **172** is coupled to non-volatile memory **175** which includes system firmware **176**. The system firmware **176** includes system BIOS **82**, which is for controlling, among other things, hardware devices in the computer system **100**. The system firmware **176** also includes ROM **180** and flash (or EEPROM) **184**. The expansion bus controller **172** is also coupled to expansion memory **188** having RAM, ROM, and/or flash memory (not shown). The system **100** may additionally include a memory module **190** that is coupled to the bus controller **112**. In one embodiment, the memory module **190** comprises a ROM **192** and flash (or EEPROM) **194**.

As is familiar to those skilled in the art, the computer system **100** further includes an operating system (OS) and at least one application program, which in one embodiment, are loaded into system memory **124** from mass storage device **152** and launched after POST. The OS may include any type of OS including, but not limited or restricted to, DOS, Windows™ (e.g., Windows 95™, Windows 98™, Windows NT™), Unix, Linux, OS/2,OS/9, Xenix, etc. The operating system is a set of one or more programs which control the computer system's operation and the allocation of resources. The application program is a set of one or more software programs that performs a task desired by the user.

In accordance with the practices of persons skilled in the art of computer programming, the present invention is described below with reference to symbolic representations of operations that are performed by computer system **100**, unless indicated otherwise. Such operations are sometimes referred to as being computer-executed. It will be appreciated that operations that are symbolically represented include the manipulation by CPU **104** of electrical signals representing data bits and the maintenance of data bits at memory locations in system memory **124**, as well as other processing of signals. The memory locations where data bits are maintained are physical locations that have particular electrical, magnetic, optical, or organic properties corresponding to the data bits.

When implemented in software, the elements of the present invention are essentially the code segments to perform the necessary tasks. The program or code segments can be stored in a processor readable medium or transmitted by a computer data signal embodied in a carrier wave over a transmission medium or communication link. The "processor readable medium" may include any medium that can store or transfer information. Examples of the processor readable medium include an electronic circuit, a semiconductor memory device, a ROM, a flash memory, an erasable ROM (EROM), a floppy diskette, a CD-ROM, an optical disk, a hard disk, a fiber optic medium, a radio frequency (RF) link, etc. The computer data signal may include any signal that can propagate over a transmission medium such as electronic network channels, optical fibers, air, electromagnetic, RF links, etc. The code segments may be downloaded via computer networks such as the Internet, Intranet, etc.

FIG. **3** illustrates a logical diagram of computer system **100**. Referring to FIGS. **2** and **3**, the system firmware **176** includes software modules and data that are loaded into system memory **124** during POST and subsequently executed by the processor **104**. In one embodiment, the system firmware **176** includes a system BIOS module **82** having system BIOS handlers, hardware routines, etc., a ROM application program interface (RAPI) module **84**, an initial start-up application (ISUA) module **86**, an initial payload **88**, cryptographic keys **90**, a cryptographic engine **92**, and a display engine **94**. The aforementioned modules and portions of system firmware **176** may be contained in ROM **180** and/or flash **184**. Alternatively, the aforementioned modules and portions of system firmware **176** may be contained in ROM **190** and/or flash **194**. The RAPI **84** provides a secure interface between ROM application programs and system BIOS **82**. The RAPI **84**, ISUA **86**, and initial payload **88***a* may each be separately developed and stored in the system firmware **176** prior to initial use of the computer system **100**. In one embodiment, the RAPI **84**, ISUA **86**, and initial payload **88** each includes proprietary software developed by Phoenix Technologies, Ltd. One embodiment of RAPI **84** is described in co-pending U.S. patent application Ser. No. 09,336,889 entitled "System and Method for Securely Utilizing Basic Input and Output System (BIOS) Services," filed on Jun. 18, 1999, assigned to Phoenix Technologies, Ltd., and which is incorporated herein by reference. One embodiment of ISUA **86** is described in co-pending U.S. patent application Ser. No. 09/336,289 entitled "Method and Apparatus for Automatically Installing and Configuring Software on a Computer," filed on Jun. 18, 1999, assigned to Phoenix Technologies, Ltd., and which is incorporated herein by reference.

In one embodiment, as shown in FIGS. **3** and **4**A and **4**B, after power is initially turned on to a new computer system **100**, the system commences with POST procedures. During the initial POST, the ISUA **86** is transferred to the mass storage device **152**, as shown by A1. In one embodiment, such a transfer is made during the manufacturing and/or assembly process, when the system **100** is first powered up after the operating system has been installed (but prior to loading and running the operating system). In an alternative embodiment, such a transfer may be made after the manufacturing and/or assembly process, after the user receives and powers up the system **100**. In a further alternate embodiment, during the transfer of the ISUA **86**, additional programs, applications, drivers, data, graphics and other information may also be transferred (for example, from ROM) to the mass storage device **152**. For example, the transfer may include the transfer of the initial payload **88***a* to the mass storage device **152**, subsequent to which the initial payload is delivered from the mass storage device **152**. Alternatively, the initial payload may be delivered from the ROM. One embodiment of the system and process for facilitating such a transfer is described in co-pending U.S. patent application Ser. No. 09/336,067 entitled "System and Method for Transferring an Application Program from System Firmware to a Storage Device" filed on Jun. 18, 1999, which is assigned to Phoenix Technologies, Ltd., the contents of which are incorporated herein by reference. Alternative embodiments of the system and process for facilitating such a transfer are described in co-pending U.S. patent application Ser. No. 09/272,859, entitled "Method and Apparatus for Providing Memory-based Device Emulation" filed on Mar. 19, 1999, in co-pending U.S. patent Continuation-in-Part application Ser. No. 09/272,859, entitled "Method and Apparatus for Providing Memory-Based Device Emulation" filed on Jun. 18, 1999, and in co-pending U.S. patent application Ser. No. 09/336,281, entitled "System and Method for Inserting One or More Files Onto Mass Storage" filed Jun. 18, 1999, each of which is assigned to Phoenix Technologies, Ltd., the assignee of the present invention, the contents of each of which are incorporated herein by reference.

In one embodiment, the ISUA **86** is a computer software executable program that will determine if there are preinstalled programs that are resident on the end user's system. If so, it will identify those preinstalled programs and create shortcuts (on the desktop in the case of a Windows operating system), or bookmarks, to allow the user to automatically launch the programs. In this embodiment, the executable program is also capable of initiating and establishing two-way communications with one or more applications on the server **22** and/or any one of the service computers **26** (FIG. **1**), as described below. Moreover, in one embodiment, graphical content of the initial payload **88***a* is displayed by display engine **94** on the user's display screen **148** during POST. Alternatively, the graphical content of the initial payload **88***a* may be displayed after a subsequent booting process. For example, as part of the user's profile as described below, the user may be asked if he or she would like to obtain additional information regarding one or more products and/or services. If the user so desires, content regarding the desired products and/or services will be displayed during subsequent boot processes.

Once POST is completed, the OS is loaded, executed, and initialized. Standard OS drivers and services are then loaded. The user is then prompted to enter registration information including demographic information such as age, gender, hobbies, etc. In addition, the ISUA **86** is executed, and runs in the background, remaining idle until it detects a communication link established between the computer system **100** and a remote server (e.g., server **22** of FIG. **1**) over Network **164** of FIG. **2** (e.g., over the Internet). In one embodiment, the ISUA **86** may search through the operating system to determine if there are applications that have been pre-loaded and pre-installed onto the system. If so, the ISUA **86** may automatically provide short cuts and/or bookmarks for the applications to launch into a predetermined server once the communication link is establish ed. This communication link can be established with a network protocol stack, (e.g. TCP/IP) through sockets, or any other two-way communications technique known in the art. Once the communication link **30** is established, the ISUA **86** issues a request signal to the server **22** (as shown by A2) to download an initial content package **62** from a content module **60**. Responsive

to the request, the server downloads the initial content package **62** (as shown by A3), which, in one embodiment, is stored in the mass storage device **152**. In one embodiment, the initial content **62** and subsequent content **64** may be developed separately, and each is encrypted and/or digitally signed using encryption keys, prior to storing of the initial content **62** and subsequent content **64** on the server **22**. When the initial content **62** and/or subsequent content **64** is/are subsequently downloaded into system **100**, the crypto engine **92** will use keys **90** to decrypt the initial content **62** and/or subsequent content **64**.

As discussed earlier, the initial content package **62** may include applications **62***a*, drivers **62***b*, and payloads **62***c*. In one embodiment, the applications **62***a* include a data loader application and a profile manager application. The data loader application functions in the same or a similar manner as ISUA **86**, and once downloaded, disables and replaces the ISUA **86**. More specifically, the data loader application is a computer software program which is also capable of initiating, establishing, and terminating two-way communications between the server **22** and the computer system **100**. The data loader application also provides traffic control management between the server **22** and computer system **100**, as well as other functions to facilitate communication between the end user's system and the designated server, and content downloading to the end user's system.

The profile manager obtains the user and system profiles of the computer system **100** based on user preferences, system hardware, and software installed at the computer system **100**. Upon obtaining the user and system profile of the computer system **100**, the profile manager application forwards the results to the data loader application, which subsequently provides the information to the server **22**, which matches the user indicted preferences with database **24** (FIG. 1). The results may be forwarded at predetermined intervals or at the user's request. The server **22** then processes the user profile or demographic data and targets content to the users which have similar profiles. In addition, the user profile data of a plurality of users are compiled on the server **22** and aggregated to create an aggregate user profile model. Content is then transmitted to user computer system's based on the user profile data and/or the aggregate user profile model (as shown by A4). The subsequent content **64** is downloaded and stored in system firmware **176**, designated by numeral **88***b*. In one embodiment, the subsequent content **64** is stored in non-volatile memory such as flash or EEPROM, with the loading of the subsequent content being done by reflashing the ROM, as is well known by those skilled in the art. The subsequent content **64** may also be stored as one or more files on mass storage device **152** or may be used to modify the Windows™ system file (under the Windows™ environment). The profile collection process is continued as long as the computer system **100** is activated. In one embodiment, content may be downloaded after the user's profile is received and analyzed at the server **22**.

When the computer system **100** is subsequently powered up (see FIG. 4B), the system again performs POST. The content that was previously downloaded and stored in system firmware **176**, and subject to copyright issues being resolved, is then displayed, prior to loading and/or execution of the operating system. In the Windows™ environment, the Windows™ logo, which is displayed during the initial loading of the operating system, is subsequently replaced by one or more screen that display the previously downloaded content stored in system firmware **176**.

In the case of storing the content as one or more files on the mass storage device **152**, as opposed to reflashing the ROM, the Windows™ logo file, which is displayed during boot-up and shutdown, may be altered or replaced. The boot-up Windows display file is named LOGO.SYS and is usually located in the Windows directory. First the Windows™ LOGO.SYS file is transferred from the Windows directory to another directory. Then, the content graphics file is renamed as LOGO.SYS and is transferred to the Windows™ directory. The operating system retrieves this file when the operating system is first launched, and hence the content is displayed on the display screen. Windows™ expects the LOGO.SYS file to be a bit-mapped file with resolution 320×400 and 256 colors although Windows™ will later stretch the resolution to 640×400 for displaying purposes. Therefore, the content graphics file is to be the same graphics format (usually named with the extension ".BMP" before being renamed to LOGO.SYS).

The operating system is then loaded, executed, and initialized. The standard operating system drivers and applications are also loaded. The profile manager is then executed. When a link has been established with the predetermined web site, additional content may be downloaded and subsequently displayed. Such additional content are either provided arbitrarily or provided based on the information obtained from a survey of the user or the user's system. In one embodiment, once the boot process is completed, a portion of the display screen may be used to provide icons or shortcuts that are used to access detailed information regarding the previously displayed messages or advertisements. In a further embodiment, the messages or advertisements may again be displayed during the shut-down process, for example, replacing the screen display that displays the message "Windows is shutting down" or "It is now safe to turn off your computer" with other selected content.

DETAILED DESCRIPTION

FIG. **5** is a diagram illustrating an architecture **500** to display an image during a transition of the operating system according to one embodiment of the invention. The architecture **500** includes a root directory **510**, a system directory **520**, a temporary directory **530**, system files **522**, **524**, and **526**, a boot-up graphic file **532**, and a shutdown graphic file **534**.

The root directory **510** is typically the C:\ drive in the mass storage where the operating system is located. The system directory **520** is typically the directory that stores the operating system that is loaded into the system memory when the BIOS boots up. In one embodiment, the operating system is the WINDOWS operating system and the system directory or folder has the name Windows.

When the BIOS loads the Windows operating system, a input/output program is executed (IO.SYS which is the historical name "input/output" known to people of the trade) and the IO.SYS attempts to locate and load the system files in a default directory, e.g., root directory and Windows. The LOGO.SYS system file **522** is used when the OS is booted up. The LOGO.SYS typically contains a image file that displays the Windows start-up logo.

When the system is shutdown, the Windows operating system retrieves the LOGOW.SYS system file **524** and the LOGOS.SYS system file **526**, and displays the appropriate logo images on the screen **148**. The LOGOW.SYS system file **524** typically contains the message "Please wait while your computer shuts down". The LOGOS.SYS system file **526** typically contains the message "It is now safe to turn off your computer".

The boot-up graphic file **532** is the file containing an image that is to be displayed during the OS boot-up in place

of the LOGO.SYS system file **522**. The shutdown graphic file **534** is the file containing an image that is to be displayed during the OS shuts down in place of the LOGOW.SYS system file **524** and the LOGOS.SYS system file **526**.

FIG. **6** is a flowchart illustrating a process **600** to display an image during a transition of the operating system according to one embodiment of the invention.

Upon START, the process **600** creates a boot-up graphic file and a shutdown graphic file using a bitmap format (Block **610**). Then the process **600** locates and retrieves the original LOGO.SYS, LOGOW.SYS and LOGOS.SYS system files from a system directory and saves them in a temporary directory under different extensions or names so that they can be used later (Block **620**).

Then the process renames the created boot-up and shutdown graphic files to LOGO.SYS for boot-up and LOGOW.SYS or LOGOS.SYS for shut-down (Block **630**). These renamed files are then transferred to the system directory where the original system files were located (Block **640**). The process **600** is then terminated.

In another embodiment, the image to be displayed during boot-up and shutdown can be captured from the graphics memory. The replacement of the Windows startup and shutdown screens has its application within the context of displaying useful information during the Windows OS startup and shutdown or subsequent startups and shutdowns on a PC.

During the firmware initialization (e.g., BIOS POST), a graphics engine generates an image into the standard Video Graphics Adaptor (VGA) as soon the adapter hardware is initialized. The resolution used is 320x400 with 256 colors. The data used to generate the image is contained with the BIOS flash memory. This data is updated regularly by an external program from within the OS.

After the BIOS has initialized the hardware of the hard drive, the graphical data (e.g., palette and bitmap information) is then stored onto the hard drive using code that supports the File Allocation Table (FAT) **16** file system format. This code is independent of the OS but offers the same functionality for writing to the hard drive as the OS. Before writing to the hard drive the code checks the compatibility of the file system and the type of the OS. The location and name of the file in the file system where the graphical data is stored corresponds to the Windows LOGO.SYS file. This file is also stored into the Windows boot directory. The exact path of the Windows directory is extracted from the MSDOS.SYS text file within the root directory of the boot drive. This file contains among other information the string "WinDir=" followed by a directory path. The filenames used under that directory are LOGOW.SYS and LOGOS.SYS.

In Windows95 the first file that is loaded after the boot-block of the hard drive is IO.SYS. This file looks for some file system compression drivers before loading the Windows LOGO graphics file from the root directory of the boot drive. IO.SYS switches the VGA into graphics mode with a resolution of 320x400 and 256 colors. It then loads the logo.sys file that contains a standard Windows Bitmap image. When Windows95 shuts down it displays two other images that are found in the Windows directory under the names LOGOW.SYS and LOGOS.SYS. Windows displays these two images in sequence with no visible interruption between them. Having the same image stored in both files gives the impression of having a single image being displayed.

The pseudo code for this process is as follows:

1. Call graphic engine to render data into the VGA card memory.
2. Identify filesystem and OS on the harddrive. If unsupported types go to step **8**.
3. Call screen capture routine to extract image from VGA card and write the image into a temporary Windows Bitmap file.
4. Locate LOGOW.SYS, and LOGOS.SYS on the file system.
5. Copy the temporary Windows Bitmap file into LOGO.SYS that is in the root directory of the boot drive.
6. Copy the temporary Windows Bitmap file into LOGOW.SYS and LOGOS.SYS into the previously located directory.
7. Delete the temporary BMP file
8. Continue with the machine booting process (finish hardware initialization and load OS).

Any error encountered during this process would cause the machine to proceed with the normal booting process in step **8**.

FIG. **7** is a flowchart illustrating a process **700** to display an image during a transition of the operating system according to one embodiment of the invention.

Upon START, the process **700** renders graphic data on the graphics memory in the graphics controller or the video display adapter (Block **710**). This rendering may be performed as part of a pre-boot graphics display activity, or as part of a normal graphics rendering process after the system boots up. The process **700** identifies the file system and the operating system on the hard drive (Block **715**). Then the process **700** determines if the file system or the OS is supported by the technique (Block **720**). If not, the process **700** is terminated.

If the file system or the OS is supported, the process **700** extracts the image from the graphic memory (Block **725**). This can be achieved by a number of techniques. One simple technique is to use a screen capture program to capture the graphics memory. Then the process **700** writes the extracted image into a temporary file having a format compatible with the operating system (e.g., bitmap) (Block **730**). In one embodiment, the operating system is a Windows-compatible OS and the file format is a bitmap format with a resolution of 320x400.

Next, the process **700** locates the system file(s) corresponding to the image or images displayed during the boot-up of shut-down (Block **735**). For example, if the OS is a Window-compatible OS, the boot-up system file is LOGO.SYS, and the shut-down system files are LOGOW.SYS and LOGOS.SYS. Then the process **700** copies the temporary bitmap file as created in block **730** to the system directory that stores the corresponding boot-up or shut-down system file (Block **740**). Then the process **700** deletes the temporary file, if necessary (Block **745**). Next, the process **700** continues the booting process and loading of the operating system (Block **750**). Note that if this process is performed at times other than booting up, then activities in block **750** are normal activities of the system. Then the process **700** is terminated.

Thus, the present invention is an efficient technique to display an image during a transition of the operating system. For boot-up, the LOGO.SYS system file is replaced by a boot-up graphic file. For shutdown, the LOGOS.SYS and/or LOGOW.SYS is replaced by a shutdown graphic file. The technique allows displaying images other than the logos from Windows.

While this invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various modifications of the illustrative embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.

What is claimed is:

1. A method to display an image during a transition of an operating system in a computer system, the method comprising:

obtaining the image having an image format compatible with the operating system; and,

creating content of a system file using the image, the system file to be accessed during the transition of the operating system, said image to correspond to a user profile.

2. The method of claim 1 further comprising:

saving the system file in a directory.

3. The method of claim 1 wherein the operating system is compatible with a Windows operating system.

4. The method of claim 3 wherein the image format is a bitmap format.

5. The method of claim 4 wherein the image has a resolution compatible with the operating system.

6. The method of claim 1 wherein the transition is a boot-up sequence.

7. The method of claim 6 wherein the system file is a LOGO.SYS file.

8. The method of claim 1 wherein the transition is a shut-down sequence.

9. The method of claim 8 wherein the system file is one of a LOGOW.SYS file and a LOGOS.SYS file.

10. The method of claim 2 wherein the directory is located on a storage compatible with the operating system.

11. A computer program product, comprising:

a computer usable medium having computer program code embodied therein to display an image during a transition of an operating system in a computer system, the computer program product having:

computer readable program code for obtaining an image having an image format compatible with the operating system; and

computer readable program code for creating content of a system file using the image, the system file to be accessed during the transition of the operating system, said image to correspond to a user profile.

12. The computer program product of claim 11 further comprising:

computer readable program code for saving the system file in a directory.

13. The computer program product of claim 11 wherein the operating system is compatible with a Windows operating system.

14. The computer program product of claim 13 wherein the image format is a bitmap format.

15. The computer program product of claim 14 wherein the image has a resolution compatible with the operating system.

16. The computer program product of claim 11 wherein the transition is a boot-up sequence.

17. The computer program product of claim 16 wherein the system file is a LOGO.SYS file.

18. The computer program product of claim 11 wherein the transition is a shut-down sequence.

19. The computer program product of claim 18 wherein the system file is one of a LOGOW.SYS file and a LOGOS.SYS file.

20. The computer program product of claim 12 wherein the directory is located on a storage compatible with the operating system.

21. A computer data signal embodied in a carrier wave comprising:

a graphic display code segment to display an image during a transition of an operating system in a computer system, the graphic display code segment comprising:

an image obtaining code segment for obtaining an image having an image format compatible with the operating system; and

a content creation code segment for creating content of a system file using the image, the system file to be accessed during the transition of the operating system, said image to correspond to a user profile.

22. The computer data signal of claim 21 wherein the graphic display code segment further comprising:

a save code segment for saving the system file in a directory.

23. The computer data signal of claim 21 wherein the operating system is compatible with a Windows operating system.

24. The computer data signal of claim 23 wherein the image format is a bitmap format.

25. The computer data signal of claim 24 wherein the image has a resolution compatible with the operating system.

26. The computer data signal of claim 21 wherein the transition is a boot-up sequence.

27. The computer data signal of claim 26 wherein the system file is a LOGO.SYS file.

28. The computer data signal of claim 21 wherein the transition is a shut-down sequence.

29. The computer data signal of claim 28 wherein the system file is one of a LOGOW.SYS file and a LOGOS.SYS file.

30. The computer data signal of claim 22 wherein the directory is located on a storage compatible with the operating system.

31. A system comprising:

a processor; and

a memory coupled to the processor, the memory containing program code to display an image during a transition of an operating system, the program code when executed by the processor causing the processor to:

obtain the image having an image format compatible with the operating system, and

create content of a system file using the image, the system file to be accessed during the transition of the operating system, said image to correspond to a user profile.

32. The system of claim 31 wherein the program code when executed by the processor further causing the processor to:

save the system file in a directory.

33. The system of claim 31 wherein the operating system is compatible with a Windows operating system.

34. The system of claim 33 wherein the image format is a bitmap format.

35. The system of claim 34 wherein the image has a resolution compatible with the operating system.

36. The system of claim 31 wherein the transition is a boot-up sequence.

37. The system of claim 36 wherein the system file is a LOGO.SYS file.

38. The system of claim 31 wherein the transition is a shut-down sequence.

39. The system of claim 38 wherein the system file is one of a LOGOW.SYS file and a LOGOS.SYS file.

40. The system of claim 32 wherein the directory is located on a storage compatible with the operating system.

41. The method of claim 1, wherein said image corresponds to data stored in a BIOS memory.

42. The method of claim 41, wherein said data is updated during execution of said operating system.

43. The method of claim 1, wherein creating content of a system file comprises creating content of a system file during said transition using the image.

44. The method of claim 1 wherein creating content of a system file comprises creating content of a system file before said transition using the image.

45. The method of claim 31, wherein said program code further causes said processor to generate said image corresponding to data stored in a BIOS memory, said image to be generated during the transition.

* * * * *

US006546489B1

(12) **United States Patent**
Frank, Jr. et al.

(10) **Patent No.: US 6,546,489 B1**
(45) **Date of Patent: Apr. 8, 2003**

(54) **DISK DRIVE WHICH PROVIDES A SECURE BOOT OF A HOST COMPUTER SYSTEM FROM A PROTECTED AREA OF A DISK**

(75) Inventors: **Charles W. Frank, Jr.**, Irvine, CA (US); **Thomas D. Hanan**, Mission Viejo, CA (US)

(73) Assignee: **Western Digital Ventures, Inc.**, Lake Forest, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/262,952**

(22) Filed: **Mar. 4, 1999**

(51) **Int. Cl.**[7] **G06F 11/30**
(52) **U.S. Cl.** **713/187**; 713/188; 713/189; 713/190; 713/193
(58) **Field of Search** 713/1, 2, 188, 713/189, 190, 193, 187

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,022,077	A		6/1991	Bealkowski et al.
5,136,713	A		8/1992	Bealkowski et al.
5,287,519	A	*	2/1994	Dayan et al. 713/202
5,341,421	A	*	8/1994	Ugon 380/52
5,377,264	A	*	12/1994	Lee et al. 713/189
H1414	H	*	2/1995	Borgen 380/21
5,430,865	A	*	7/1995	Lazik 713/2
5,432,939	A	*	7/1995	Blackledge, Jr. et al. ... 713/200
5,432,950	A	*	7/1995	Sibigtroth 713/193
5,712,973	A	*	1/1998	Dayan et al. 713/200
5,802,069	A		9/1998	Coulson
5,809,337	A		9/1998	Hannah et al.
5,835,760	A		11/1998	Harmer
5,844,986	A		12/1998	Davis
6,185,507	B1	*	2/2001	Huber et al. 702/30

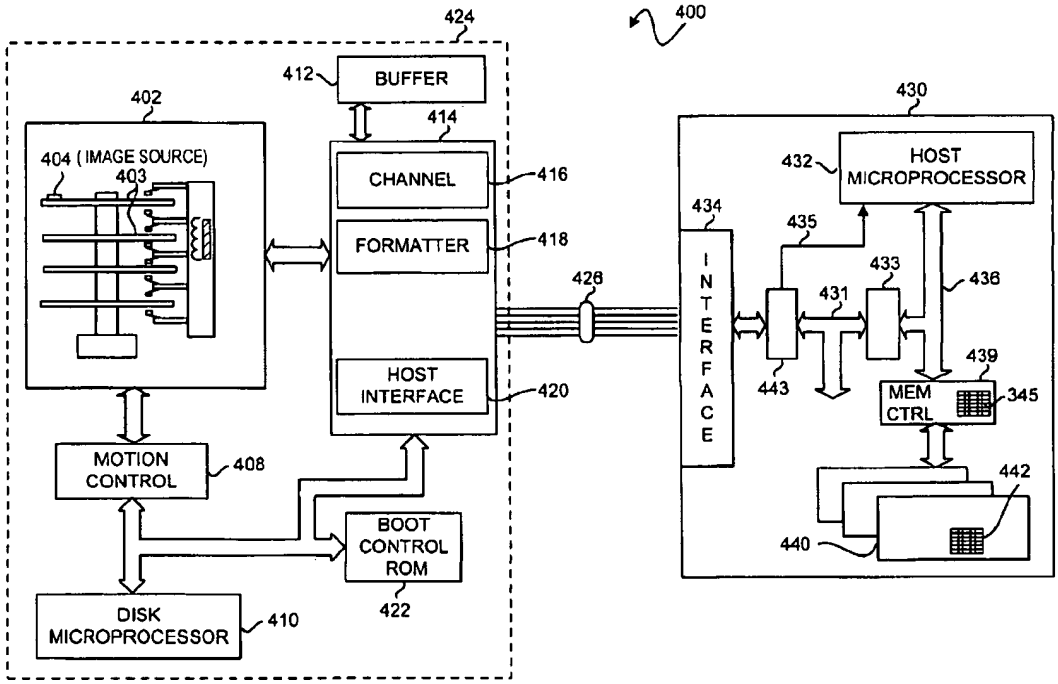
* cited by examiner

Primary Examiner—Thomas R. Peeso
Assistant Examiner—Kambiz Zand
(74) *Attorney, Agent, or Firm*—Milad G. Shara, Esq.; Myers Dawes & Andras; Ramin Mobarhan, Esq.

(57) **ABSTRACT**

A computer system comprises a host computer having a memory array and a host microprocessor, and a disk drive having a drive microprocessor. The disk drive provides a secure boot load of the host computer by causing the host microprocessor to remain in an inactive state while a template for loading host computer memory is read by a drive microprocessor from a protected area of the disk and loaded into host memory via the host interface. The host computer may then be activated with a memory image source whose source is impervious to virus attack or inadvertent corruption. A method is disclosed for creating and updating the secure template. The host interface may be an I/O interface or a memory referenced interface.

16 Claims, 8 Drawing Sheets



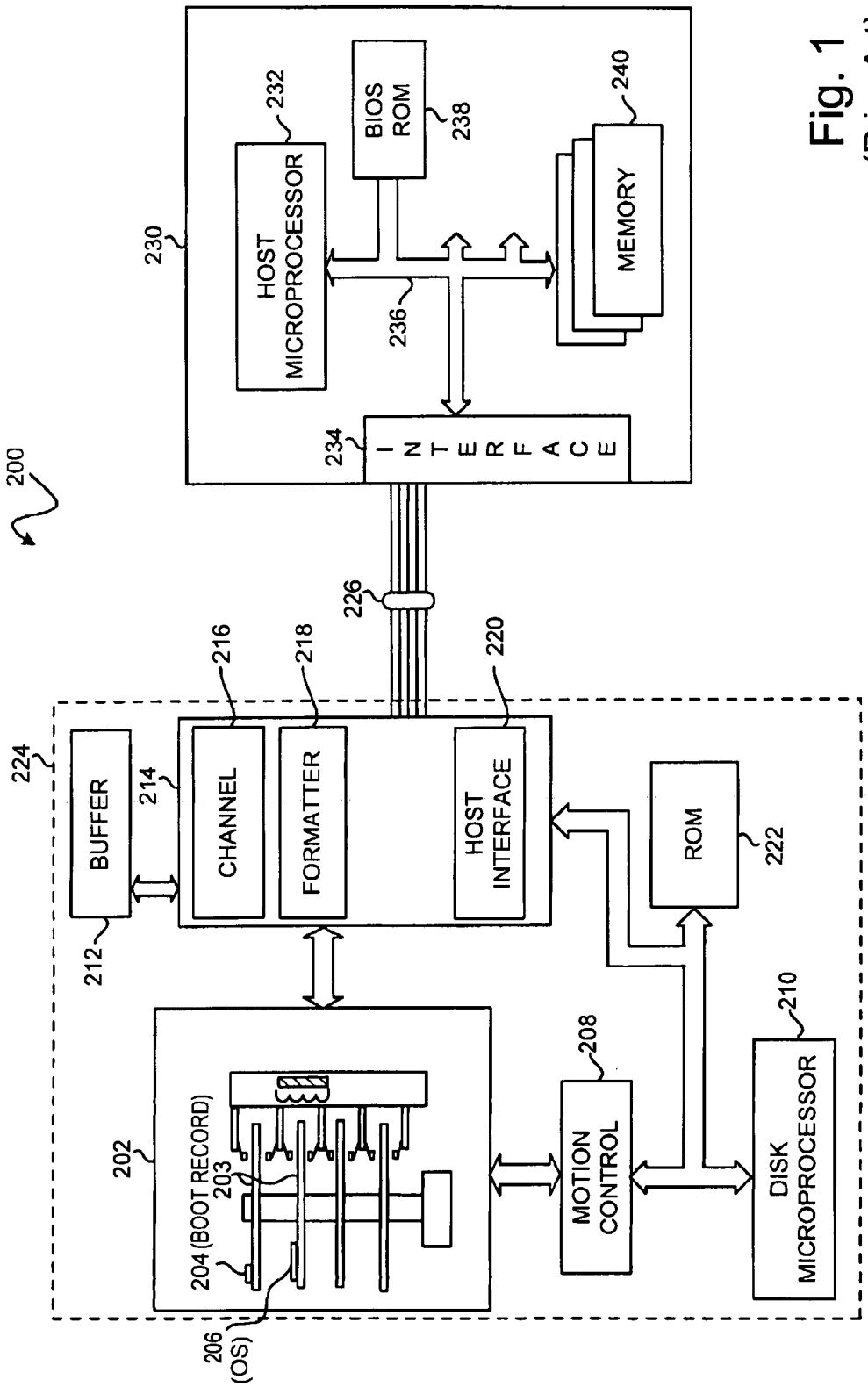


Fig. 1
(Prior Art)

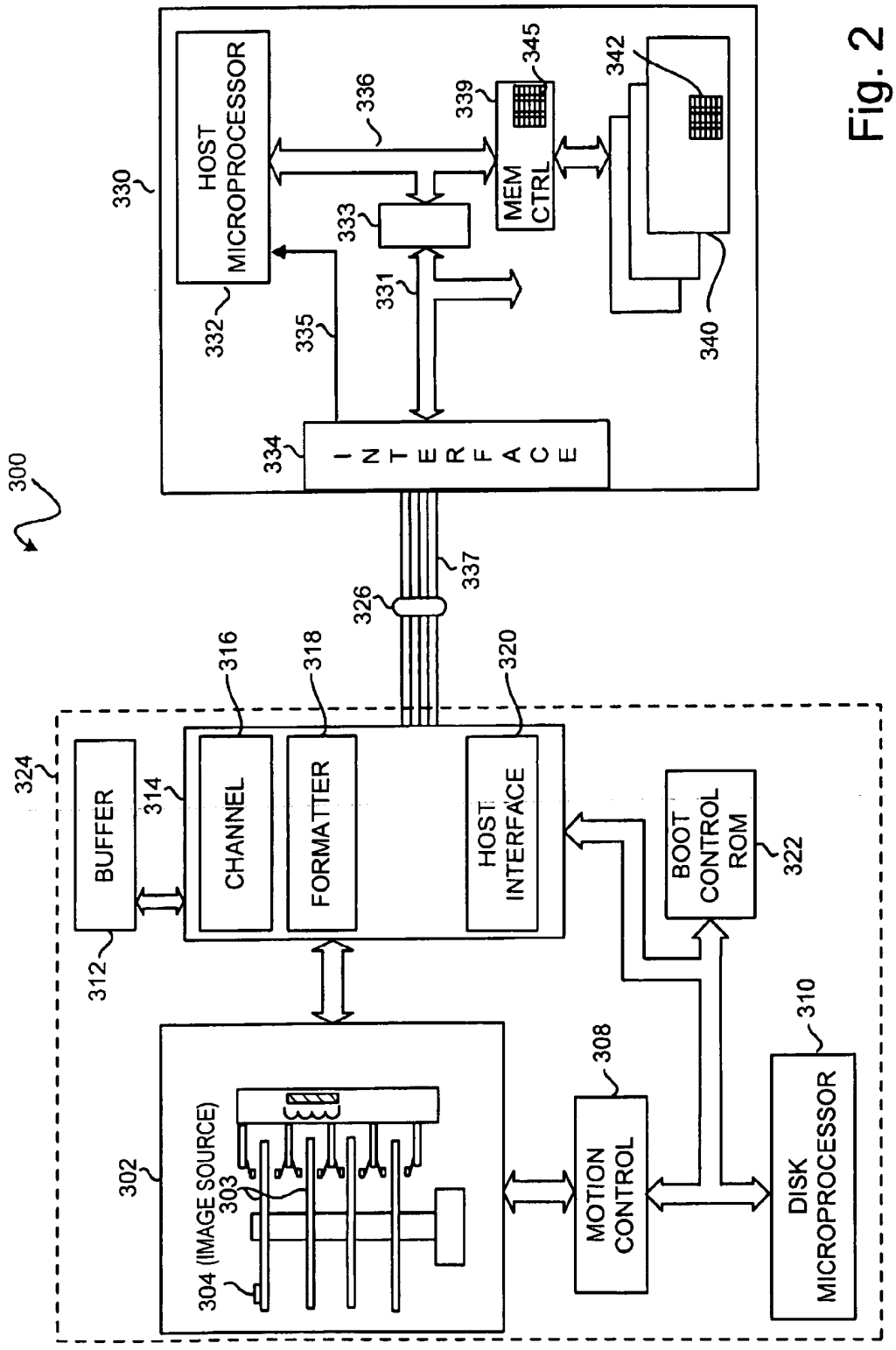

300
330
HOST MICROPROCESSOR
332
336
339
345
MEM CTRL
342
340
333
331
335
334
INTERFACE
337
326
316
318
320
324
BUFFER
314
CHANNEL
FORMATTER
HOST INTERFACE
312
BOOT CONTROL ROM
322
302
304 (IMAGE SOURCE)
303
308
MOTION CONTROL
310
DISK MICROPROCESSOR

Fig. 2

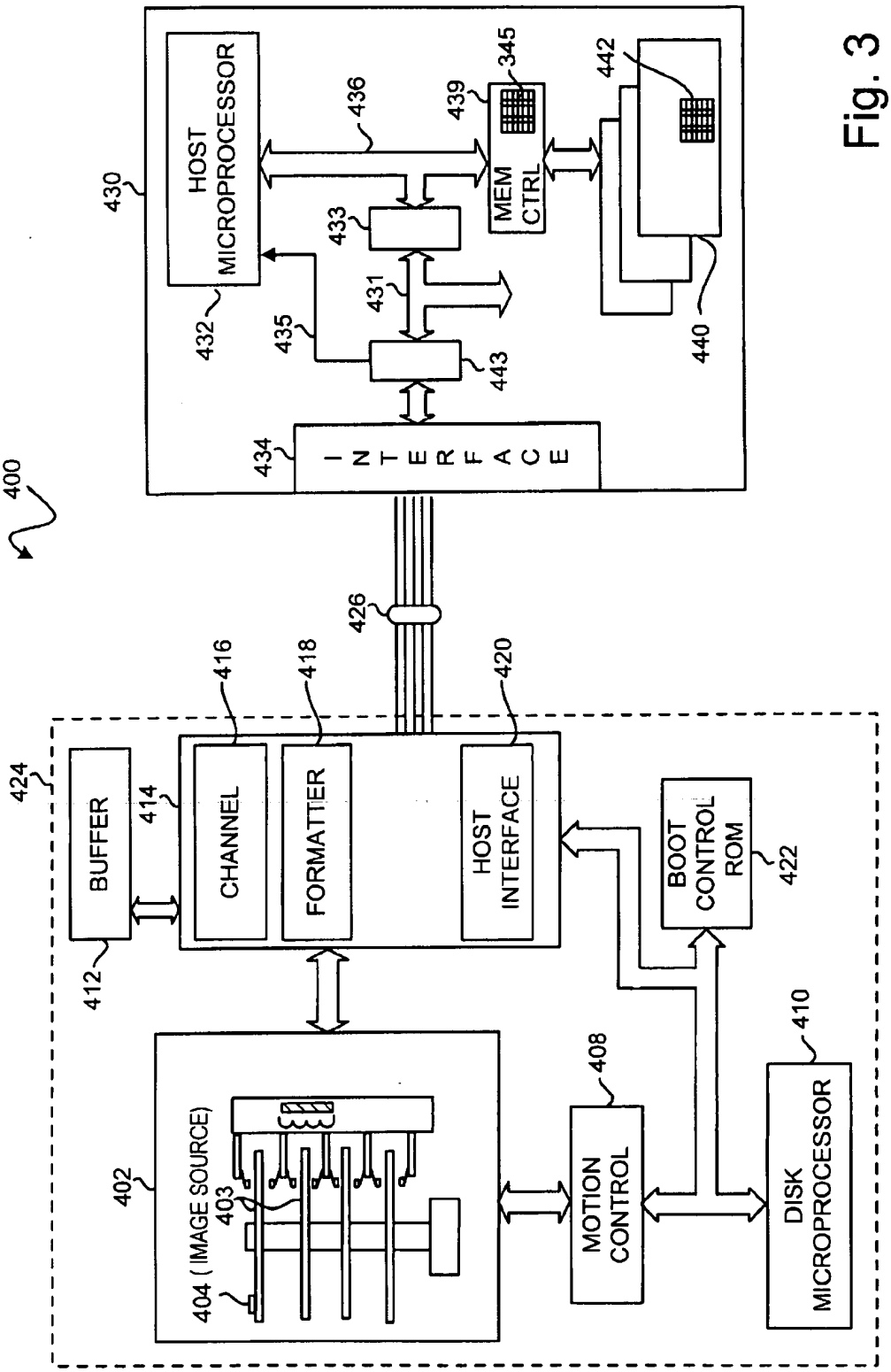

400
430
HOST MICROPROCESSOR
432
436
439
345
MEM CTRL
442
433
431
435
443
440
434
INTERFACE
426
416
418
420
424
BUFFER
414
CHANNEL
FORMATTER
HOST INTERFACE
412
BOOT CONTROL ROM
422
408
410
402
403
404 (IMAGE SOURCE)
MOTION CONTROL
DISK MICROPROCESSOR

Fig. 3

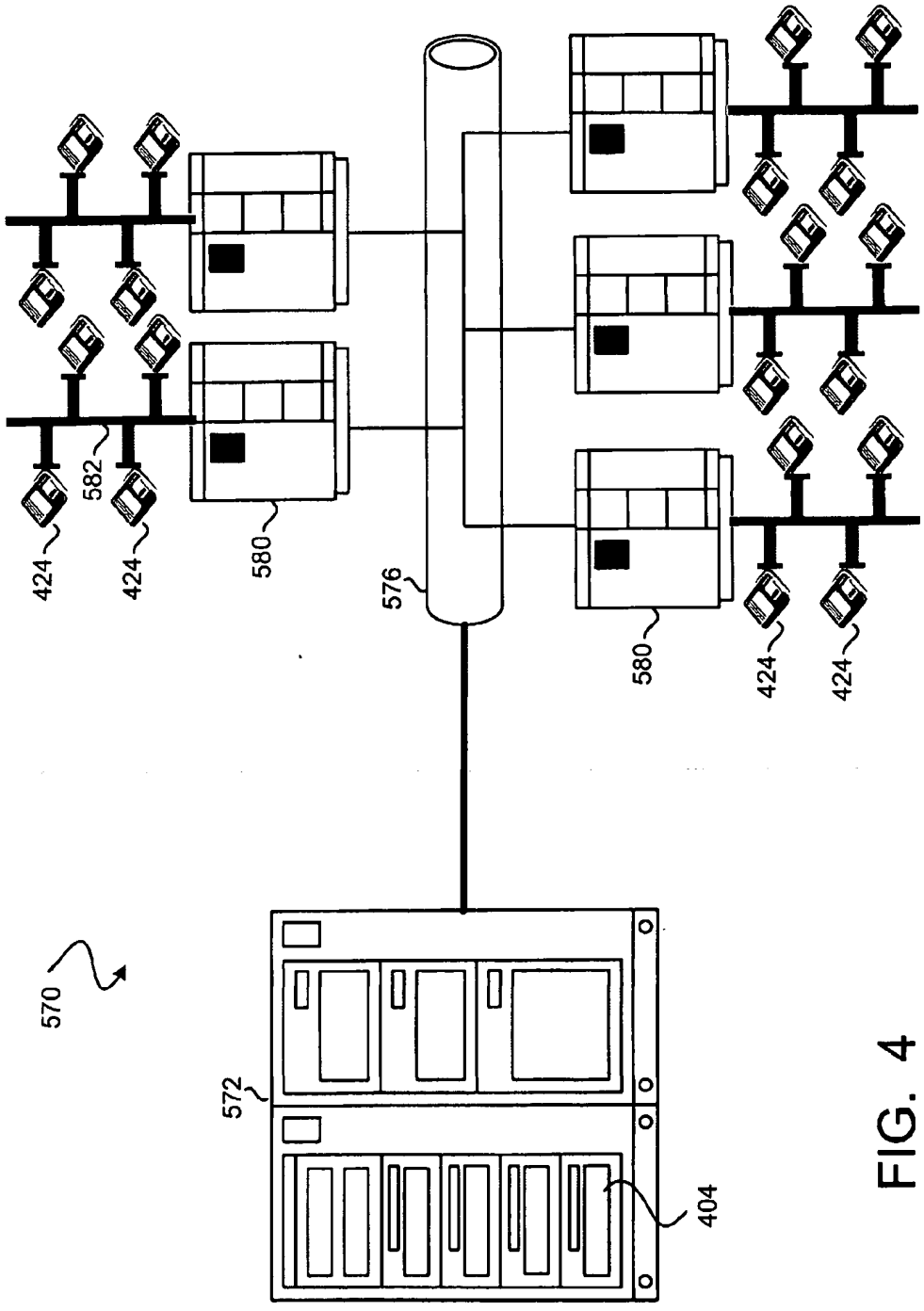


FIG. 4

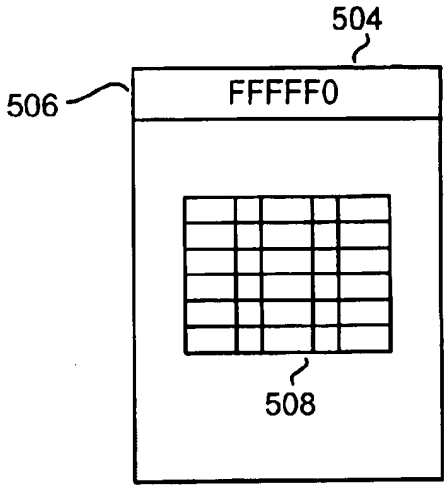

504
506
FFFFF0
508

FIG. 5

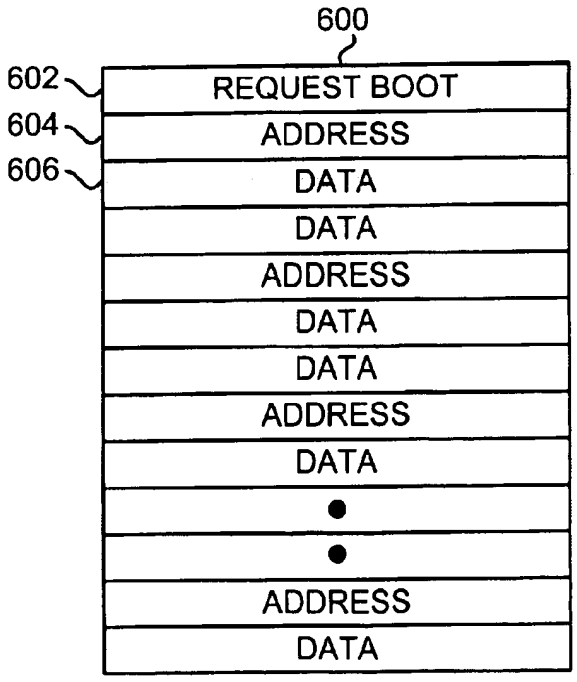

600
602
604
606
REQUEST BOOT
ADDRESS
DATA
DATA
ADDRESS
DATA
DATA
ADDRESS
DATA
ADDRESS
DATA

FIG. 6

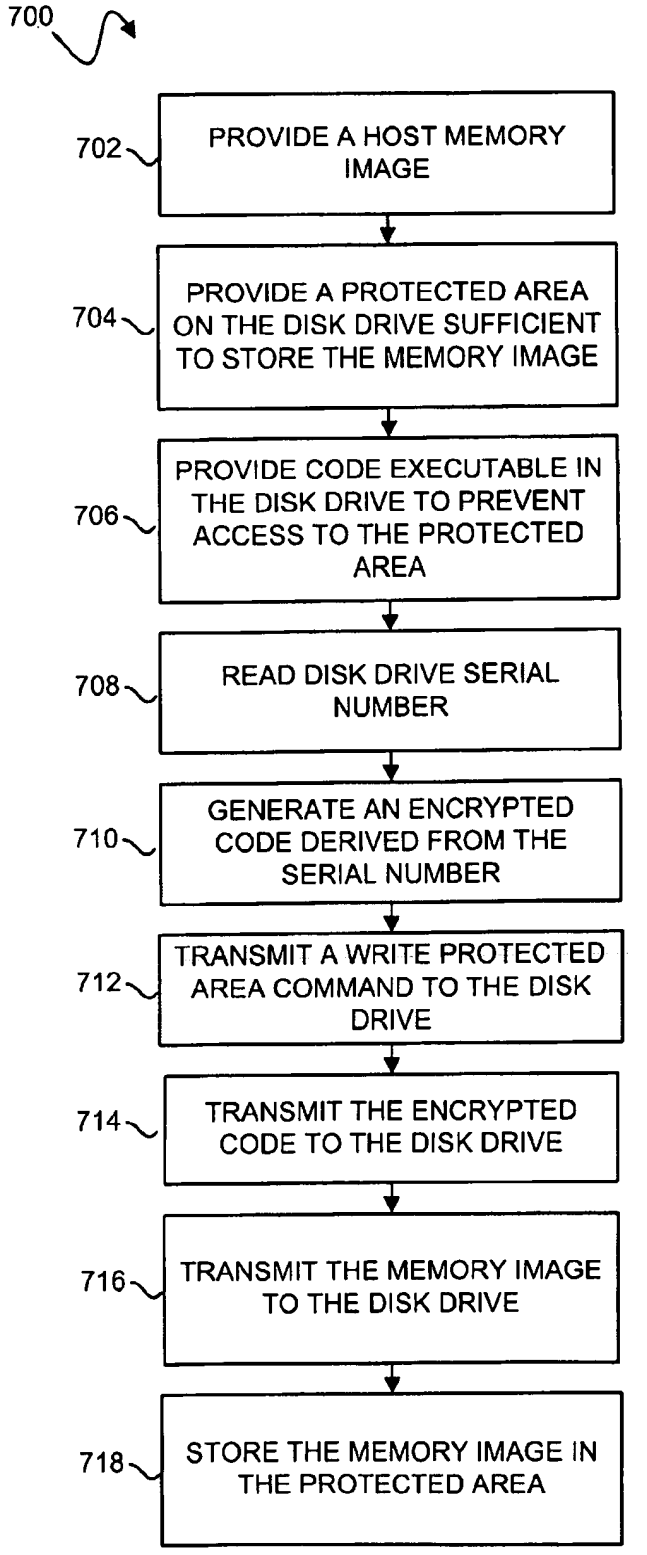

700
702
PROVIDE A HOST MEMORY IMAGE
704
PROVIDE A PROTECTED AREA ON THE DISK DRIVE SUFFICIENT TO STORE THE MEMORY IMAGE
706
PROVIDE CODE EXECUTABLE IN THE DISK DRIVE TO PREVENT ACCESS TO THE PROTECTED AREA
708
READ DISK DRIVE SERIAL NUMBER
710
GENERATE AN ENCRYPTED CODE DERIVED FROM THE SERIAL NUMBER
712
TRANSMIT A WRITE PROTECTED AREA COMMAND TO THE DISK DRIVE
714
TRANSMIT THE ENCRYPTED CODE TO THE DISK DRIVE
716
TRANSMIT THE MEMORY IMAGE TO THE DISK DRIVE
718
STORE THE MEMORY IMAGE IN THE PROTECTED AREA

FIG. 7

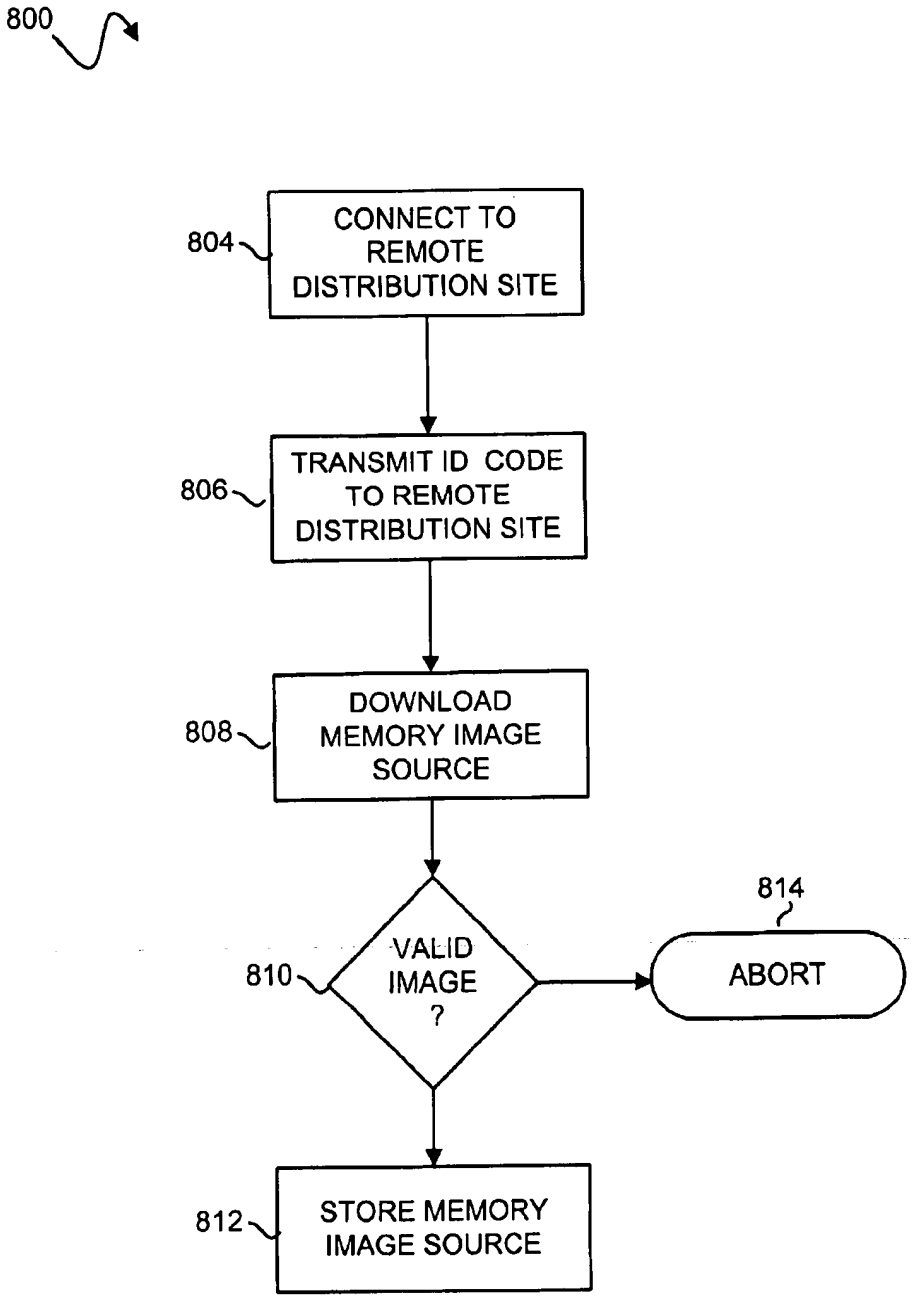

800
804
CONNECT TO REMOTE DISTRIBUTION SITE
806
TRANSMIT ID CODE TO REMOTE DISTRIBUTION SITE
808
DOWNLOAD MEMORY IMAGE SOURCE
810
VALID IMAGE ?
814
ABORT
812
STORE MEMORY IMAGE SOURCE

FIG. 8

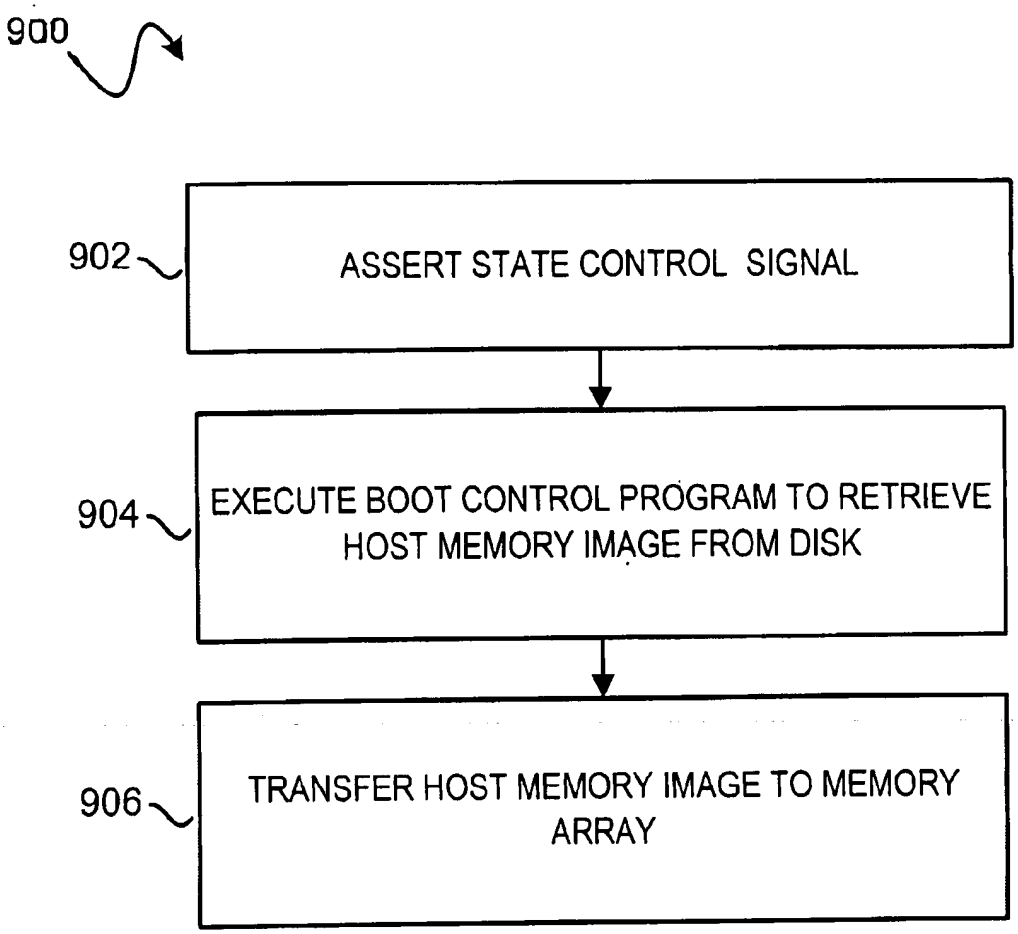

900
902
ASSERT STATE CONTROL SIGNAL
904
EXECUTE BOOT CONTROL PROGRAM TO RETRIEVE HOST MEMORY IMAGE FROM DISK
906
TRANSFER HOST MEMORY IMAGE TO MEMORY ARRAY

FIG. 9

DISK DRIVE WHICH PROVIDES A SECURE BOOT OF A HOST COMPUTER SYSTEM FROM A PROTECTED AREA OF A DISK

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to the field of secure boot loading of a computer system from a hard disk drive. In particular, the invention relates to a source for fast restoration of a complete operating image in computer system memory which is secure from attack by a virus or inadvertent corruption during operation of the computer.

2. Description of the Prior Art and Related Information

Most computer systems today take the form of a so-called "personal computer" or PC which has evolved into a ubiquitous tool applied in many forms. Examples include desktop systems, servers, and "embedded" systems which incorporate a PC as the engine for performing dedicated functions. Common to most of these systems is a host microprocessor and a disk drive. The host microprocessor executes program code, including operating system code and application program code, and reads or writes data in conjunction with code execution. The code and associated data is stored for execution in a volatile random access memory array. The disk drive provides non-volatile secondary storage for the code and data. The extent of disk drive storage is orders of magnitude greater than the memory array, allowing numerous application programs, and potentially a plurality of operating systems, to reside on the disk drive for recall according to dynamic configurations of the machine.

The memory array is initially loaded during a bootstrap (boot) loading process which begins with the host microprocessor executing a relatively small BIOS program stored in a ROM. The BIOS program reads a default area of the disk which stores a boot program, known as a boot record, and stores the program in the memory array. The host microprocessor then executes the boot program to load an operating system core which may then complete the process of establishing an operating image in memory.

Unfortunately, the PC is susceptible to problems during this process. Computer viruses are rampant, many of which plant themselves in the boot record or in the operating system code on the disk so that they may be activated during operation of the machine. Other forms of computer viruses simply corrupt or destroy code on the disk which prevents the machine from booting up at all. Aside from virus attacks, it is possible that inadvertent corruption of disk data can prevent a proper boot of the computer system. This can be caused by user mistakes or by rogue applications which fail to abide by conventions or operating system safeguards.

Many tactics have been employed to defend the data on the disk drive from virus attack. One method was to provide BIOS code which monitored disk drive write commands to look for attempted boot record modification. This and similar BIOS-based methods depend on virus software employing BIOS calls to access the disk and therefore may be ineffective when a virus bypasses BIOS. Another known method employs bus snooping hardware which monitors the I/O bus to trap disk write operations to protected areas. All these methods are prone to defeat because the host processor is required to access the data and may be controlled by a virus.

In another aspect, it is known in the art to provide an abridged version of BIOS in ROM and use the ROM BIOS to load the full BIOS from the disk drive or some other alterable memory. Since the BIOS itself is susceptible to attack or corruption in these implementations, there have been efforts to provide protection. One such a system is disclosed in U.S. Pat. No. 5,022,077 to Bealkowski et al. Bealkowski discloses having the host processor send a command to the disk drive after a BIOS is loaded to establish a maximum block address. The BIOS code is stored on the disk at addresses which are higher than the maximum block address and are therefore inaccessible until the maximum block address is reset. This method also presents the requirement that the host processor controls the protection scheme and the protection method can be easily defeated.

A more complex BIOS protection scheme is disclosed in U.S. Pat. No. 5,844,986 to Davis. The Davis patent discloses a cryptographic coprocessor which acts as a gatekeeper to BIOS stored in a flash memory. The cryptographic coprocessor responds to BIOS addresses presented by the host microprocessor during BIOS reads and requires decoding an encrypted code to process updates to the BIOS. The Davis patent provides a solution to BIOS security but adds cost from flash memory and an additional processor, and does not address potential contamination of operating system code. Further, Davis admits that an intruder can corrupt the code if the secret key is obtained.

Yet another problem experienced by PC users is the time required to perform the boot load process. The operating system code on the disk drive is a complex arrangement of linked blocks which are loaded in many stages with considerable processing required. In addition, most complex operating systems require a previous orderly shut-down to achieve an efficient start-up. Unfortunately, the orderly shut-down is sometimes as lengthy as the boot process. One known solution to the boot load delay, sometimes known as "resume from disk" or "hibernation," has been to store the system memory image in special partition on the disk drive. A subsequent start-up operation retrieves the image and resumes at the prior state of the machine. This solution is advantageous when starting the machine, but still presents a significant shut-down delay. Further, the image on disk is susceptible to virus attack or corruption as noted above.

There is a continuing need, therefore, for a computer system boot process which is fast and secure from virus attack or inadvertent corruption.

SUMMARY OF THE INVENTION

This invention can be regarded as a computer system comprising a host computer, a disk drive, and means defining a host interface between the host computer and the disk drive. The host computer comprises a host microprocessor having an inactive state and an active state. The host microprocessor has an input for receiving a state-control signal and while the state-control signal is asserted remains in the inactive state, and while the state-control signal is de-asserted remains in the active state. While in the active state, the host microprocessor executes host-executable code including operating system and application program code. The host computer further comprises a memory array for storing the host-executable code and data, means coupled between the memory array and the host interface for reading from and writing to the memory array; and means responsive to a signal on the host interface for asserting and de-asserting the state-control signal.

The disk drive comprises a disk having disk addresses for storing and retrieving data including data defining a host

computer memory image source, means for storing and retrieving drive-executable code including code defining a boot control program, and a drive microprocessor for executing the drive-executable code including the boot control program.

The host computer memory image source is stored at disk addresses which are accessible by the drive microprocessor when executing the boot control program and which are protected from access by the host computer. The host computer memory image source further comprises an address pointer to establish an address in the memory array for storing at least a portion of the memory image source.

The computer system further comprises means for transferring the host computer memory image source to the memory array via the host interface while the drive microprocessor is executing the boot control program means controlled by the drive microprocessor for causing the state-control signal to be asserted.

The invention may be used with a host interface which is either a memory referenced interface or an I/O interface.

In another aspect, the invention may be viewed as a method for providing a secure boot load image in a computer system comprising a disk drive and a host computer. The method comprises the steps of providing a host memory image source; providing a protected area of the disk sufficient to store the host memory image source; providing an encrypted code; providing code executable in the disk drive to prevent access to the protected area by the host computer unless the protected area command and the encrypted code is sent to the disk drive by the host computer; transmitting the protected area command and the encrypted code to the disk drive; transmitting the host memory image source to the disk drive; and storing the host memory image source in the protected area.

Preferably the encrypted code is derived from the disk drive serial number. The image source may be stored as a contiguous image or as a compressed image.

In another aspect, the step of providing a host memory image source may include the steps of connecting to a remote distribution site; transmitting an identification code which uniquely identifies the computer system to the remote distribution site; downloading the host memory image source from the remote distribution site; and validating the host memory image source.

In still another aspect, the invention can be summarized as a method for securely booting the aforementioned computer system. The method comprises the steps of asserting the state control signal; executing the boot control program with the drive microprocessor to retrieve a host memory image source from the disk drive; and while the boot control program is executed and the state-control signal is asserted, transferring the host memory image to the memory array.

The foregoing and other features of the invention are described in detail below and set forth in the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a prior art computer system with a BIOS ROM in the host computer and a conventional disk drive storing a boot record and an operating system.

FIG. **2** is a computer system according to an embodiment of this invention employing a memory-referenced host interface between the disk drive and the host computer.

FIG. **3** is a computer system according to another embodiment of this invention employing an I/O host interface between the disk drive and the host computer, and providing a coprocessor on the host computer for cooperating with the disk drive microprocessor to cause the host microprocessor to be inactivated and a secure host memory image to be loaded into memory.

FIG. **4** is block diagram of a test system which is suitable for loading the secure host memory image source onto the disk drive.

FIG. **5** is a representation of the host memory image source stored on disk with an address pointer and the image template.

FIG. **6** is a representation of the protocol for transmitting the host memory image over a host interface which is an I/O interface such as IDE or SCSI.

FIG. **7** is a flow chart showing the method of the invention for initially providing a host memory image and storing it in a protected area of the disk drive, such as during manufacturing of the disk drive.

FIG. **8** is a flow chart showing an alternate embodiment of the method of the invention for accessing a remote site to provide an updated host memory image and store it in a protected area of the disk drive.

FIG. **9** is a flow chart showing the method of the invention for providing a secure boot of a host computer from a protected area of the disk drive.

DETAILED DESCRIPTION

FIG. **1** shows a prior art computer system **200** comprising a disk drive **224** and a host computer **230**. A host interface **226** is defined between host computer **230** and disk drive **224** which is conventionally an IDE (sometimes known as ATA) or SCSI interface. Various forms of the IDE or SCSI interface, complying with particular specifications, are in use which provide different levels of performance and function.

Disk drive **224** comprises a head disk assembly (HDA) **202** and a set of controller integrated circuits **214** which may be integrated in various forms. HDA **202** comprises one or more rotating disks **203** (4 shown) mounted on a spindle motor and a moveable head stack assembly having head transducers for accessing data on the disks. The spindle motor and the head stack assembly are controlled by a motion control circuit **208** which provides current drivers and control logic. A channel **216** provides signal processing including encoding and decoding for data transferred to and from the head transducers. A formatter **218** provides block level digital processing of disk data and may include error correction and detection logic. A buffer **212** provides temporary storage of data being read from or written to the disk and may be implemented in form of a cache memory. A host interface **220** provides logic and drivers to respond to host interface **226**. A drive microprocessor **210** executes code to control disk operations and manage a queue of commands from the host. A ROM **222** stores initialization code executed by drive microprocessor **210**.

Host Computer **230** comprises a host microprocessor **232**, a BIOS ROM **238**, a memory array **240**, and a host interface circuit **234** which drives and responds to host interface **226**. Bus **236** connects host computer **232** to the aforementioned elements. In this simplified diagram, conventional components such as memory control logic or other peripheral devices are omitted, but are well known to those skilled in the art.

During a boot load process, host microprocessor **232** executes code in BIOS ROM **238** to access a boot record **204**

on disk **203** and proceeds thereupon to load an operating system image in memory from operating system source **206** stored on one or more disks **203**. As previously indicated, boot record **204** and operating system source **206** are subject to contamination by a computer virus or inadvertent modification.

FIG. **2** illustrates a computer system **300** according to an embodiment of the invention comprising host computer **330** and disk drive **324**. Host computer **330** comprises host microprocessor **332**, host local bus **336**, memory controller **339**, memory array **340**, Peripheral Component Interface (PCI) bridge **333**, local PCI bus **331** and host interface control logic **334**. Host microprocessor **332** has an active state when executing instructions, and an inactive state brought about by the assertion of a state-control signal such as a reset or hold signal, both well known in the art. In the inactive state, host microprocessor **332** is prevented from accessing memory array **340**. Host microprocessor **332** is suitably a Pentium™ class microprocessor, although other microprocessor families may be used with equal advantage. Host interface logic **334** preferably comprises a memory based interface such as a PCI expansion bus coupled to disk drive **324** via host interface bus **326** and coupled to memory array **340** via PCI bridge **333** and memory controller **339**. Other memory referenced interfaces including both serial and parallel types may be employed. The memory referenced interface between disk drive **324** and host computer **330** enables disk drive **324** to load data into memory array **340** via host interface **334**, local PCI bus **331**, PCI bridge **333**, and memory controller **339**.

Disk drive **324** comprises channel **316**, formatter **318**, motion control **308**, buffer **312** and HDA **302**, comprising disks **303**. A drive microprocessor **310** executes a disk control program to initialize the disk drive. A portion of the storage capacity on disks **303** is partitioned to provide a protected area of disk addresses which are known to the disk control program, but are inaccessible to host computer **330**. The protected area is sufficient to store an image source **304** suitable to recreate a fully functional operating image in memory **340**. When computer system **300** is initialized, such as following a power-up sequence, host interface controller **320** asserts a state-control signal **337** which is translated in host interface **334** to assert internal state-control signal **335**, thereby causing host microprocessor **332** to be maintained in an inactive state such as reset or hold.

After state-control signal **337** is asserted, drive microprocessor **310**, executing code in boot control ROM **322**, reads a host memory image source **304** from the above-mentioned protected area of disk **303** and generates addresses and data therefrom for writing into memory array **340** via the preferred PCI interface to host computer **330**. Host interface controller **320** provides logic and buffering for interfacing between the host interface PCI bus **326** and drive microprocessor **310**. When memory array **340** has been loaded with the operating image from host memory image source **304**, state-control signal **337** is de-asserted, thereby allowing host microprocessor **332** to resume an active state and begin executing the host-executable code stored in memory array **340**.

In one embodiment, a portion of host memory image source **304** comprises a BIOS code set. To ensure the security of the BIOS code set, disk microprocessor **310** uses the memory referenced access path described above to store the BIOS code in a portion **342** of memory array **340**, and writes to registers **345** in memory controller **339** to write-protect the portion **342** of memory array **340** from being overwritten. Preferably, one or more of the registers **345** stores a code which must be provided to memory controller **339** in order to enable portion **342** to then be overwritten after the protection is established.

The just described process provides an efficient and fully secure boot load of computer system **300**. There is no requirement for code to be executed by host microprocessor **332** during the restoration of an operating image in memory array **340**. Consequently there is no requirement for a BIOS ROM in host computer **330** and overall no opportunity for a virus to contaminate the operating image stored on disk. The time required to restore the operating image may be significantly shorter than prior art boot loads or even resume from disk operations, because no intervening processing is necessary.

Turning to FIG. **5**, a diagram of one embodiment **504** of the host image source stored on disk is shown. An address pointer **506** provides a starting memory address location in memory array **340** to begin loading data. Following address pointer **506**, a contiguous block of data **508** is provided representing the host memory image. Numerous embodiments of host image source **304** are possible within the scope of the invention including compressed images, non-contiguous images with interspersed address pointers and encrypted images.

FIG. **3** shows an alternate embodiment of the invention where an I/O interface is used to connect a host computer and a disk drive. Computer system **400** comprises host computer **430** and HDA **424**. In general, elements in FIG. **3** are comparably numbered with FIGS. **1** and **2** (e.g. HDA's **202,302,402**) so that only those elements which are most relevant to the invention need be discussed.

Host interface **426**, supported by host interface controller **420** within disk drive **424** and interface control logic **434** in host computer **430**, is preferably an IDE interface. A SCSI or other standard I/O interface may alternatively be used. When the invention is used with an I/O interface instead of a memory referenced interface such as PCI, some intervening control logic must be employed to address memory array **440**. A boot load micro-controller **443** in host computer **430** monitors signals from host interface logic **434** for a boot request from disk drive **424**, typically following a power-up or system reset sequence. Upon recognizing that a boot load sequence is in progress, micro-controller **443** asserts state-control signal **435** to cause host microprocessor **432** to enter an inactive state. Subsequently, micro-controller **443** receives boot load address and data information from disk drive **424** and writes the data into memory array **440** at the indicated addresses. Upon completion of the boot load, state-control signal **435** is de-asserted by micro-controller **443** and host microprocessor **432** returns to an active state and executes the program just loaded.

In order to write protect a portion **442** of memory array **440** comparable to the process discussed above for FIG. **2**, microcontroller **443** receives register data from disk microprocessor **410** and writes the data into registers **445** in memory controller **439**.

FIG. **6** shows a sequence **600** of data which may be communicated by disk drive **424** to micro-controller **443** during the boot load process. Sequence **600** comprises a request boot code **602** which is recognized by micro-controller **443** to assert state-control signal **435**. Subsequently a stream of address **604** and data words **606** may be transmitted to transmit the host memory image source to memory array **440**.

FIG. **9** summarizes the method of the invention **900** to perform a secure boot load of a computer system. In step

902, the state-control signal is asserted by the drive microprocessor to cause the host microprocessor to enter an inactive state. In step **904**, the drive microprocessor executes a boot control program to retrieve the host memory image from disk. The method proceeds to step **906** where the drive microprocessor transfers the host memory image to the memory array.

FIG. **4** shows a system **570** which is suitable for manufacturing disk drives with a pre-loaded host memory image source in a protected area of the disk. System **570** comprises a mainframe or central computer system **572**, a plurality of disk drive test systems **580** (5 shown) and a plurality of disk drives **424** connected to the disk drive test systems via host interfaces **582**. A network **576** provides a communication link between mainframe **572** and the plurality of disk drive test systems **580**. In principle, manufacturing system **570** is similar to the system disclosed in commonly assigned pending U.S. patent application Ser. No. 08/873,230, the disclosure of which is hereby incorporated by reference. Mainframe **572** maintains a copy of host operating image source **404** in its internal storage bay and provides the copy to each test system **580** for transmittal to disk drives **424**. Each disk drive **424** is assigned a bar coded serial number upon its introduction to test system **580** and thus is able to form an unique encrypted code which is preferably derived from its serial number. Since the drive serial number and the algorithm used for generating the encrypted code are known to the test system **580**, the drive can be induced to accept a write operation to its protected area. The algorithm may also take into account other parameters known only to system **570** and the disk drive, and these other parameters may be employed later in the disk drive's life to enable an update of the host memory image source after leaving the factory. System **570** provides for a record of the encrypted code for this use, and further provides sufficient capacity for simultaneously manufacturing numerous versions of disk drives or disk drive based systems with various unique host memory image source files.

FIG. **7** illustrates a preferred method **700** of the invention for providing the host memory image source to the disk drive and storing it thereupon. In step **702** a host memory image is provided as discussed above. In step **704**, the disk drive provides a protected area sufficient to store the host memory image source. In step **706**, the drive is provided with code executable in the disk drive to prevent access to the protected area unless an enabling command and code sequence is received. In step **708** the disk drive serial number is obtained. In step **710**, an encrypted code which is at least partially derived from the disk drive serial number is computed. In step **712**, a command to write in the protected area is transmitted to the disk drive. In step **714**, the encrypted code is transmitted to the disk drive. In step **716**, the host memory image source is transmitted to the disk drive. Finally in step **718**, the host memory image source is stored in the disk protected area, having been enabled by transmitting the special command and the encrypted code. Preferably an additional algorithm is employed which in which the host memory image source includes some form of self-verification which may be appending syndrome or CRC bytes or other methods which ensure that the image is valid.

FIG. **8** shows an alternate embodiment **800** of the method step of providing a host memory image source which may be applied to update the image after the drive has been installed in a user's computer system. In step **804**, the computer system is connected a remote distribution site such as the manufacturer's Internet web site. In step **806**, the computer system transmits an ID code to the remote distribution site. In step **808**, the host memory image source is downloaded. In step **810**, the image is validated by the disk drive. If the image is not valid, the process is aborted at step **814**, otherwise a valid image is stored in the protected area at step **812**.

We claim:

1. A computer system comprising:

a host computer;

a disk drive;

means defining a host interface between the host computer and the disk drive;

the host computer comprising:

a host microprocessor having an inactive state and an active state, the host microprocessor having an input for receiving a state-control signal and while the state-control signal is asserted remaining in the inactive state, and while the state-control signal is de-asserted remaining in the active state and therein executing host-executable code including operating system and application program code;

a memory array for storing the host-executable code and data;

means coupled between the memory array and the host interface for reading from and writing to the memory array;

means responsive to a signal on the host interface for asserting and de-asserting the state-control signal;

the disk drive comprising:

a disk having disk addresses for storing and retrieving data including data defining a host computer memory image source;

means for storing and retrieving drive-executable code including code defining a boot control program;

a drive microprocessor for executing the drive-executable code including the boot control program;

the host computer memory image source being stored at disk addresses which are accessible by the drive microprocessor when executing the boot control program and which are protected from access by the host computer;

the host computer memory image source further comprising an address pointer to establish an address in the memory array for storing at least a portion of the memory image source;

means for transferring the host computer memory image source to the memory array via the host interface while the drive microprocessor is executing the boot control program; and

means controlled by the drive microprocessor for causing the state-control signal to be asserted.

2. The computer system of claim **1** wherein the host interface is a memory-referenced interface.

3. The computer system of claim **2** wherein the memory-referenced interface is a PCI bus.

4. The computer system of claim **2** wherein the means for transferring the host computer memory image source comprises bus mastering circuits.

5. The computer system of claim **1** wherein the host interface is an I/O interface.

6. The computer system of claim **5** wherein the means coupled between the memory array and the host interface comprises a boot load microcontroller which is operable while the state-control line is asserted.

7. The computer system of claim **6** further comprising a means for the boot load microprocessor to detect a boot request sequence on the host interface.

8. A method for securely booting a computer system comprising a disk drive and a host computer, the host computer being coupled to the disk drive via a host interface; the disk drive comprising a drive microprocessor, a boot control program executable by the drive microprocessor and a means for causing a state-control signal to be asserted; the host computer further comprising a memory array and a host microprocessor, the method comprising:

asserting the state-control signal from the disk drive to hold the host microprocessor in an inactive state wherein the host microprocessor is prevented from accessing the memory array when the state control signal is asserted;

executing the boot control program with the drive microprocessor to retrieve a host memory image from the disk drive; and

while the boot control program is executed and the state-control signal is asserted, transferring the host memory image to the memory array.

9. The method of claim 8, further comprising:

de-asserting the state-control signal from the disk drive to restore the host microprocessor to an active state wherein the host microprocessor is allowed to access the memory array when in an active state; and

accessing the transferred host memory image from the memory array by the host microprocessor.

10. The method of claim 9, wherein the host memory image is retrieved from an area of the disk drive protected from access by the host computer.

11. The method of claim 8,

wherein the disk drive further comprises a disk,

wherein host memory image is retrieved from a protected area of the disk sufficient to store the host memory image, and

wherein a code executable in the disk drive prevents the host computer from accessing to the protected area unless a protected area command and a predetermined encrypted code are sent to the disk drive by the host computer.

12. The method of claim 11, further comprising:

transmitting the protected area command and the encrypted code to the disk drive to enable writing data in the protected area;

transmitting the host memory image to the disk drive; and

storing the host memory image in the protected area prior to the asserting.

13. The method of claim 11, wherein the encrypted code is derived from a serial number of the disk drive.

14. The method of claim 11, further comprising:

compressing the host memory image prior to the storing wherein the host memory image stored in the protected area is the compressed host memory image.

15. The method of claim 8, wherein the computer system is in communication with a remote distribution site and wherein the method further comprising:

transmitting a unique identification code of the computer system to the remote distribution site;

receiving the host memory image from the remote distribution site; and

validating the received host memory image.

16. The method of claim 15, wherein the remote distribution site is in communication with the computer system via the Internet.

* * * * *